

NHIỀU TÁC GIẢ

DIJKSTRA

ẤN PHẨM CHUYÊN ĐỀ CHO KỸ SƯ PHẦN MỀM NGƯỜI VIỆT

TẬP 1 | THÁNG 04 - 2018

THIẾT KẾ HỆ THỐNG QUẢN LÝ
DANH MỤC SẢN PHẨM TRONG
HỆ THỐNG ECOMMERCE tr.08

CHUYÊN ĐỀ

| **FLOATING POINT VÀ SAI SỐ
TRONG TÍNH TOÁN** tr.24

GROKKING'S CORNER

KĨ SƯ PHẦN MỀM GIỎI THÌ
PHẢI NHƯ THẾ NÀO? tr.30

grokking
VIETNAM



NHÀ XUẤT BẢN ĐỒNG NAI

Mục lục

TẬP 1 | THÁNG 04 - 2018

05 Về Grokking Vietnam

The Grokking Team



08 Thiết kế hệ thống quản lý danh mục sản phẩm trong hệ thống eCommerce

Lê Minh Nghĩa - Tiki.VN

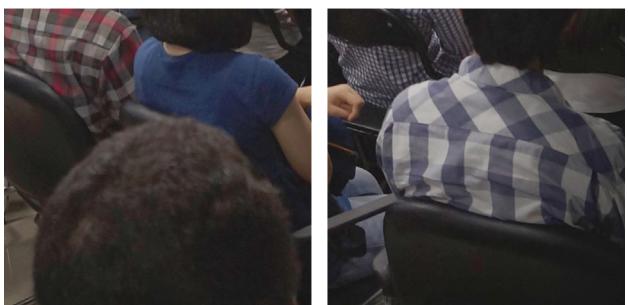
14 Lock Free Programming

Kipalog



18 HTTP/1 và HTTP/2

Lê Nguyên Dũng



22 PostgreSQL Buffer hoạt động như thế nào?

Phạm Hữu Luân

24 Floating Point và sai số trong tính toán

Phú Trần

28 Thiết kế tính năng Feature Toggle & Feature Rollout

Huy Nguyễn

30 Grokking's Corner: Kỹ sư phần mềm giỏi thì phải như thế nào?

Nguyễn Văn Quang Huy

33 Sự hình thành và phát triển của các cộng đồng công nghệ ở Việt Nam

The Grokking Team

38 Sách hay nên đọc

Grokking Việt Nam

Xây dựng hệ sinh thái cho cộng đồng kỹ sư phần mềm trình độ cao ở Việt Nam

Lời mở đầu

Bạn đọc thân mến,

Bạn đang cầm trên tay tập đầu tiên của ấn phẩm Dijkstra - ấn phẩm chuyên đề dành cho kỹ sư phần mềm người Việt - kết quả của một thời gian dài thai nghén ý tưởng của đội ngũ Grokking Vietnam.

Trong ấn phẩm này, nhóm biên soạn sẽ cố gắng cung cấp những bài viết chuyên sâu về lĩnh vực công nghệ phần mềm, những bài phân tích kiến trúc, những bài viết về văn hóa, cộng đồng và các bài viết về kiến thức nền tảng khoa học máy tính. Để đảm bảo tính chính xác cũng như tính cập nhật, những bài viết này sẽ được viết bởi chính những kỹ sư phần mềm đang làm việc trong những công ty phần mềm tại Việt Nam hoặc nước ngoài.

Trong quá trình biên soạn tờ ấn phẩm này sẽ không tránh khỏi những sơ suất, nhóm biên soạn mong nhận được những ý kiến đóng góp để nhóm biên soạn có thể không ngừng nâng cao chất lượng bài viết của ấn phẩm trong những số sau.



GROKKING VIETNAM

Grokking Vietnam

Chủ trách nhiệm xuất bản

Võ Thành Lộc

Biên tập

Phạm Thanh Thuý

Phạm Hữu Luân

Đỗ Anh Thông

Thiết kế và dàn trang

Đỗ Anh Thông

Nguyễn Thị Minh Tâm

Cộng tác nội dung

Nguyễn Văn Quang Huy

Lê Minh Nghĩa

Lê Nguyên Dũng

Phú Trần

Đỗ Xuân Huy

Phạm Hữu Luân

Khắc Anh

Cùng với sự góp ý của các bạn trong nhóm Grokking Research

Mọi bài vở, ý kiến đóng góp xin
vui lòng gửi về dijkstra@grokking.org





grokking

Thành lập năm 2014

từ 3 founders, Grokking hiện có đội ngũ nhân sự vận hành xấp xỉ 20 người và cộng đồng Slack với hơn 1000 thành viên tham gia trên cơ sở tình nguyện, và cùng hướng tới một sứ mệnh chung:

*“Xây dựng hệ sinh thái cho cộng
đồng kỹ sư phần mềm trình độ cao ở Việt
Nam”*

về GROKKING VIỆT NAM

Grok (v) /grok/ to understand something so profoundly that it fully absorbed in oneself

GROKKING TEAM

Thân chào các bạn,

Đã gần ba năm kể từ hoạt động đầu tiên (11/2014), đội ngũ Grokking Vietnam đã đi qua một chặng đường dài với nhiều dự án được tổ chức như Tech Talks, Hackathon, Training Course, ... nhưng có nhiều bạn vẫn chưa thực sự hiểu được Grokking là gì. Do đó, trong số đầu tiên của DIJKSTRA, chúng mình muốn nhân cơ hội này để giới thiệu lại về tổ chức Grokking Vietnam, về quá trình thành lập cũng như những giá trị mà Grokking Vietnam đang theo đuổi. Mong rằng từ đây, những ai cảm thấy phù hợp với những giá trị Grokking mang lại sẽ có động lực đến với chúng mình nhiều hơn, còn những ai nhìn thấy những cơ hội mà Grokking bỏ lỡ, sẽ giúp chúng mình nhận ra và phát triển để đạt tới tầm kỳ vọng của mọi người.

GROKKING – MỘT TỔ CHỨC KHÔNG VÌ LỢI NHUẬN

Bắt đầu hoạt động đầu tiên vào năm 2014, khi trên địa bàn TP HCM đã có khá nhiều cộng đồng lập trình viên sinh hoạt như JSMeetup, Ruby Meetup, Grokking thường được mọi người nhận diện như một cộng đồng công nghệ. Tuy nhiên, chúng mình muốn định nghĩa bản thân là một tổ chức không vì lợi nhuận (not for profit), tức là Grokking sẽ có *cơ cấu hoạt động, quy trình và sứ mệnh cụ thể* để tổ chức được bền vững và mang lại nhiều giá trị nhất cho giới kĩ sư phần mềm tại Việt Nam. Kể từ năm 2014 tới nay, từ một nhóm nhỏ gồm 3 sáng lập viên, Grokking hiện có đội ngũ nhân sự xấp xỉ 20 người và cộng đồng Slack với hơn 1000 thành viên. Toàn bộ nhân sự của chúng mình đều hoạt động trên cơ sở tình nguyện, và cùng hướng tới một sứ mệnh chung của Grokking:

“Xây dựng hệ sinh thái cho cộng đồng kĩ sư phần mềm trình độ cao ở Việt Nam.”

Khi nhắc tới sứ mệnh này, mỗi thành viên trong Grokking đều có thể kể một câu chuyện riêng về việc vì sao họ thực sự luôn hướng tới sứ mệnh đó. Nhưng có lẽ, câu chuyện nổi bật nhất chính là câu chuyện về việc muốn biến Việt Nam thành một Tech Hub của châu Á, chuyển dịch từ định vị trong suy nghĩ

nhiều người hiện nay là một Outsourcing Hub.

Bạn Huy - founder của Grokking đã chia sẻ: “Kỹ sư Việt Nam thực sự rất giỏi, rất lành nghề nếu tính trên phương diện outsourcing, nhưng điều làm kỹ sư Việt Nam thua xa Singapore và Mỹ là thiếu tư duy nền tảng”.

Và sau đây là những điều chúng mình tin tưởng sẽ giúp rút ngắn khoảng cách đó:



- ▶ Xây dựng hệ sinh thái cho cộng đồng kĩ sư phần mềm trình độ cao tại Việt Nam.



- ▶ Chia sẻ kiến thức kĩ thuật chuyên sâu.
- ▶ Tiếp cận bài toán kĩ thuật với góc độ ứng dụng trong thực tế.
- ▶ Rèn luyện khả năng tư duy logic và giao tiếp hiệu quả.



- ▶ TECH TALK
- ▶ RESEARCH
- ▶ NEWSLETTER
- ▶ MAGAZINE

CHIA SẺ NHỮNG KIẾN THỨC KĨ THUẬT CĂN BẢN VÀ CHUYÊN SÂU CỦA NGÀNH KHOA HỌC MÁY TÍNH VÀ KĨ SỰ PHẦN MỀM

Với niềm tin rằng trở thành một kỹ sư phần mềm giỏi, chúng ta không chỉ cần thành thục các kỹ năng lập trình mà còn phải nắm chắc những kiến thức căn bản nhất của ngành CS (Computer Science) & SE (Software Engineering), sau một thời gian bần bạc và chuẩn bị, nhóm sáng lập viên của Grokking đã tổ chức Grokking Tech Talk #1 với chủ đề về Database được tổ chức vào tháng 11 năm 2014, với gần 30 người tham dự.

Những trao đổi, thảo luận tại buổi Tech Talk đầu tiên đã chứng minh những chủ đề căn bản và mang tính học thuật chuyên sâu cũng rất được cộng đồng đón nhận. Năm 2015 có tổng cộng 4 Tech Talk được diễn ra với những chủ đề như: Build big things from small component, Real-time Data pipeline and Real-world service Architecture, Android from Linux perspective and Root from Android, Coding standard and performance

optimization, Techology stack at MySquar, ...

Con số tham dự những buổi Tech Talk của Grokking luôn ổn định và tăng dần từ 30, 40 lên tới 70, 80 bạn. Hầu hết những phản ứng chúng mình quan sát được đều rất tích cực, như mọi người nán lại hỏi nhiều, kiên nhẫn chờ câu hỏi của mình được giải đáp. Từ những sự kiện này, Grokking thực sự thấy được sự hữu ích của việc đem những kiến thức nền tảng và khoa học máy tính chia sẻ cho cộng đồng lập trình viên.

Sau đây là chia sẻ của một số người tham dự về những tác động mà Tech Talk mang lại:

"Minh thấy ở Tech Talk, kiến thức thường có hàm lượng kỹ thuật cao, kiến thức đó có được sẽ đủ để mình tìm hiểu tiếp. Ý Minh là khối kiến thức đó có thể giúp Minh tự tìm hiểu, học hỏi thêm, có thể giúp Minh giải thích, suy luận khi gặp trường hợp tương tự. Ví dụ như cái talk Progres của anh Huy, những kiến thức cơ bản khi một database vận hành giúp Minh hiểu những thao tác nào làm cho database phình to ra."

"Trước giờ dõi với mình, Grokking Tech Talk có vai trò mở rộng tầm mắt là chủ yếu. Còn áp dụng thực tiễn thì thường đến từ Ruby meetup và Full-stack weekend nhiều hơn."

"...đi để mang và có cái nhìn rõ hơn cho hướng đi của mình".

Dựa vào những lời nhận xét như vậy, cộng với việc quan sát hướng đi của các cộng đồng khác, chúng mình đi tới kết luận: để mang lại giá trị nhiều nhất, Grokking sẽ tập trung cung cấp những kiến thức kĩ thuật chuyên sâu cho các bạn lập trình viên. Điều này cũng hòa hợp với một niềm tin khác của chúng mình, đó là việc phân biệt một bạn lập trình viên lành nghề và một bạn lập trình viên giỏi sẽ nằm ở chỗ: lập trình viên giỏi sẽ không bị phụ thuộc vào một loại ngôn ngữ, công nghệ cụ thể mà còn có khả năng giải quyết bài toán bằng tư duy kĩ thuật có hệ thống.

KHÁM PHÁ CÁCH NHỮNG KIẾN THỨC KĨ THUẬT CHUYÊN SÂU ĐƯỢC ỨNG DỤNG TRONG THỰC TẾ

Một điều chúng mình nhận ra khi theo dõi các Grokking Tech Talk là: thay vì cách thể hiện lí thuyết bình thường, thì việc chia sẻ các bài toán thực tế trước rồi từ đó quay lại giải đáp những kiến thức cơ bản được ứng dụng để giải quyết những bài toán này, tư duy của người tham dự sẽ được "kích thích" nhiều hơn. Ví dụ như thế này:

Trong ứng dụng chuyển tiền của 1 ngân hàng Việt Nam, có một bạn lập trình như sau:

1. def transfer(A, B, amount):
2. DB.transaction do
3. if A.balance > amount
4. A.balance -= amount

5. A.save!
6. B.balance += amount
7. B.save!
8. end
9. end
10. end

Sau khi ứng dụng chạy được một thời gian thì quản lí hệ thống phát hiện tài khoản của vài người dùng bị âm. Bạn có thể giải thích tại sao? Và có cách nào để xử lý vấn đề này.

Khi tiếp xúc với những bài toán được mô tả dưới góc độ vận hành/ kinh doanh như thế này, các bạn kĩ sư phần mềm đòi hỏi phải tự vận động tư duy để trả lời những câu hỏi: Mô hình kinh doanh của hệ thống vận hành như thế nào? Khách hàng/ người dùng có thói quen ra sao? Nền tảng công nghệ nào có thể giải quyết được bài toán này? Những dữ kiện của bài toán này yêu cầu một hệ thống kĩ thuật có điều kiện ra sao? Muốn triển khai thì phải làm như thế nào? ... Chính những câu mà các

"Kĩ sư Việt Nam thực sự rất giỏi, rất lành nghề nếu tính trên phương diện outsourcing, nhưng điều làm kĩ sư Việt Nam thua xa Singapore và Mỹ là thiếu tư duy nền tảng." | Nguyễn Văn Quang Huy

bạn tự hỏi mình như vậy đã đem lại giá trị nhiều hơn cho các bạn khi quay lại làm việc. Vì giờ đây các bạn đã có thể giao tiếp tốt hơn với các bộ phận khác trong công ty.

RÈN LUYỆN KHẢ NĂNG TƯ DUY HỆ THỐNG VÀ GIAO TIẾP HIỆU QUẢ CHO CÁC LẬP TRÌNH VIÊN

Với mục tiêu cung cấp thêm nhiều kiến thức cho các bạn lập trình viên bối bối, luyện não nhiều hơn nữa, bên cạnh Tech Talk, chúng mình lập nên trang Facebook Page và trang engineering.grokking.org để chia sẻ với tần suất nhiều hơn. Đây chính là lúc mà Grokking nhận ra một phương án tiếp theo có thể giúp nâng cao trình độ của các bạn kĩ sư phần mềm lên đáng kể: **tập trung vào cách các bạn tư duy và giao tiếp.**

Để giải thích về điều này, chúng mình xin được kể về hoạt động xây dựng nội dung cho trang Facebook của Grokking. Những mẫu nội dung này có thể là các bài viết ngắn, hoặc phân tích dài về một chủ đề bất kì liên quan tới khoa học máy tính và kĩ sư phần mềm. Lúc đầu, các bạn trong nhóm nội dung của Grokking sẽ giúp tổng hợp những bài viết này từ trên mạng. Tất cả các bạn đều là các lập trình viên trẻ, mong muốn trau dồi kiến thức của mình thông qua hoạt động này của Grokking.

Một vấn đề này ra trong quá trình làm việc của nhóm xây dựng nội dung là thứ nhất, các bạn tuy biết về một kiến thức nào đấy nhưng lại không biết viết xuống để cho người khác hiểu dễ dàng. Các bạn thường không biết một người khác đọc bài mình sẽ nảy ra những câu hỏi gì tiếp theo để đưa ra thông tin giải đáp một cách hệ thống khiến cho họ thỏa mãn. Chúng



mình tin rằng, việc “giao tiếp không tới” này cũng có thể xuất hiện khi các bạn phải thuyết phục đồng nghiệp trên công ty, khi bàn luận với cùng bạn bè, hay khi hỏi ý kiến mentor. Tóm lại thông tin các bạn thu vào và đưa ra càng không hệ thống và phân mảnh, hiệu quả công việc sẽ càng không cao.

Để khắc phục điều này, cách hoạt động của nhóm đã được thay đổi một chút. Thay vì chỉ nghiên cứu cá nhân và viết bài, mỗi thành viên trong nhóm giờ sẽ phải thực hiện hẳn một bài trình bày trước các bạn khác, làm cho các bạn hiểu được về vấn đề mình đang nói, trả lời giải đáp thắc mắc của các bạn nếu có, sau đó mới viết lại thành bài hoàn chỉnh. Đây là một quá trình mất khá nhiều thời gian và công sức, tuy nhiên, nó khiến cho các bạn tham gia luyện tập khả năng tư duy hệ thống và giao tiếp hiệu quả. Nhờ có nó mà cách các bạn viết bài sâu hơn hẳn, không những thế, chúng mình còn quan sát thấy những dấu hiệu thay đổi tích cực ở nhóm các bạn làm nội dung: đọc nhiều hơn, hỏi nhiều hơn, chăm tới văn phòng hơn. Có bạn còn quyết định cày hết cả quyển sách về ORACLE để thuyết trình. Một bạn trong team nội dung đã chia sẻ: “Các kiến thức có được từ việc viết bài giúp mình đào sâu hơn, mở rộng những hiểu biết hiện tại, thậm chí là khi tìm kiếm về nó, còn phát hiện ra nhiều công cụ/knowledge liên quan khác.”

NHỮNG DỰ ĐỊNH TRONG TƯƠNG LAI

Sau khi đã xác định những yếu tố đem lại giá trị cốt lõi của Grokking bao gồm việc cung cấp kiến thức kĩ thuật chuyên sâu, bổ sung góc nhìn từ khía cạnh non-tech và rèn luyện khả năng tư duy, giao tiếp có hệ thống, hiện Grokking đang có 4 hoạt động chính ổn định: *Grokking TechTalks*, *Grokking Research*,

Grokking Online Magazine.

Tuy vậy, chúng mình vẫn muốn khám phá thêm những cách thức mới và hiệu quả hơn để đạt được sứ mệnh của mình. Trong tương lai, Grokking sẽ tiếp tục thử nghiệm và phát triển thêm nhiều hoạt động mới - như ấn phẩm DIJIKSTRA này là một ví dụ - để tìm ra những hoạt động tốt nhất phục vụ cộng đồng kĩ sư phần mềm tại Việt Nam.

Mỗi hoạt động mới đưa ra đều dựa trên giả thuyết, niềm tin nào đó của Grokking về cách phát triển một kĩ sư phần mềm xuất sắc, và chúng mình sẽ cần rất nhiều đóng góp, nhận xét để củng cố niềm tin, chứng minh giả thuyết và nâng cao chất lượng của các hoạt động trong tương lai.

Nếu bạn là một kĩ sư phần mềm thuộc mọi trình độ, nền tảng, hãy chia sẻ cho chúng mình biết, bạn đang cần gì và thiếu gì nhé.

Grokking cũng sẵn sàng chào đón sự hợp tác, giúp đỡ của những nhà quản lý, những doanh nghiệp hoặc tổ chức cùng quan tâm tới chất lượng của kĩ sư phần mềm Việt Nam. Hãy liên lạc với chúng mình tại email core@grokking.org nếu bạn muốn kết nối.



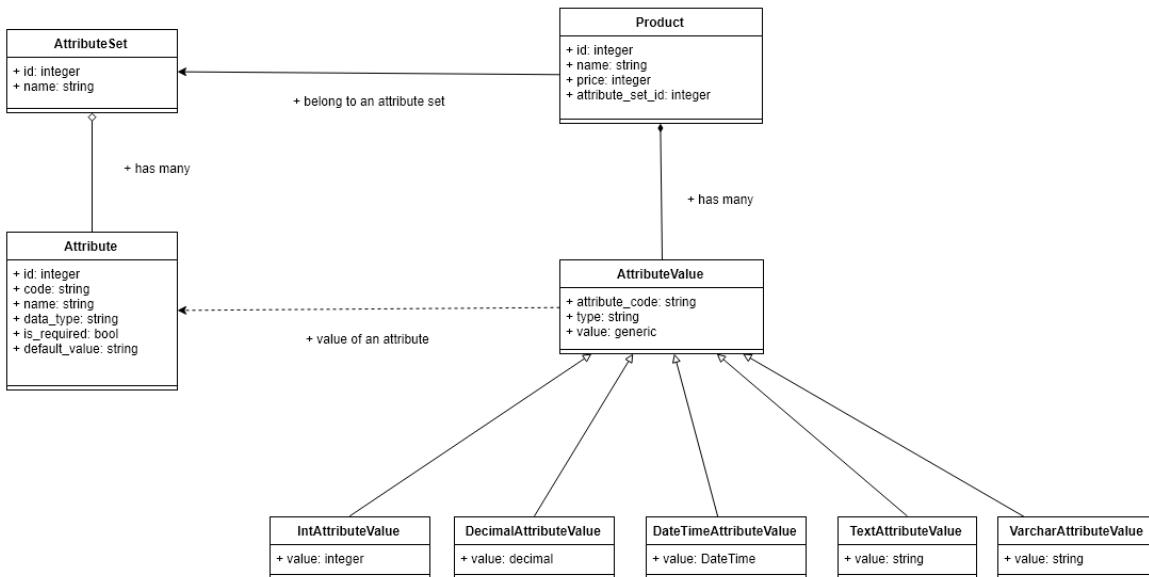
Quản lý danh mục sản phẩm (QLDMSP) là một trong các thành phần quan trọng nhất của một hệ thống eCommerce. Cùng với quản lý đơn hàng và giao vận, QLDMSP tạo thành ba trụ cột chính. Chính vì vậy việc xây dựng hệ thống QLDMSP có vai trò quan trọng trong kiến trúc tổng thể của toàn hệ thống.

thiết kế hệ thống, **quản lý** **danh mục sản phẩm** trong hệ thống **Ecommerce**

Nguồn: Internet

Lê Minh Nghĩa - Tiki.VN

Hiện là solution architect tại tiki.vn. Là một người đam mê xây dựng hệ thống đáp ứng hiệu năng cao, dễ dàng mở rộng với các nghiệp vụ phức tạp.



1- XÂY DỰNG MÔ HÌNH (MODEL) ĐỂ LƯU TRỮ THÔNG TIN VỀ SẢN PHẨM

Đối với các hệ thống eCommerce thì sản phẩm (product) là đối tượng hiển thị chính các thông tin để người dùng thực hiện việc mua bán.

Một sản phẩm có nhiều thuộc tính đa dạng về số lượng, loại thuộc tính, kiểu dữ liệu. Mỗi thuộc tính có các yêu cầu khác nhau về tính chất dữ liệu như: độ dài, kiểu kí tự, các yêu cầu về nghiệp vụ... Do đó mô hình sản phẩm (product model) là một mô hình không có cấu trúc cố định (schema-less structure).

Cách tiếp cận phổ biến để giải quyết vấn đề này là sử dụng mô hình dạng EAV - Entity Attribute Value. Đây là mô hình được Magento, Wordpress sử dụng để lưu trữ các cấu trúc đa dạng không định trước. Đặc điểm của cấu trúc EAV là tách phần lưu trữ giá trị thuộc tính (attribute value) khỏi phần cấu trúc các thuộc tính (attribute) của mô hình. Nhờ đó có thể định nghĩa được nhiều loại cấu trúc thuộc tính khác nhau của sản phẩm.

Trong sơ đồ trên, bộ thuộc tính (attribute set) là bảng dùng để lưu một tập hợp các thuộc tính. Các thuộc tính được phân biệt với nhau bởi mã thuộc tính và loại kiểu dữ liệu. Một thuộc tính có thể thuộc nhiều bộ thuộc tính khác nhau. Một sản phẩm chỉ thuộc một bộ thuộc tính. Các giá trị thuộc tính (attribute value) của sản phẩm tương ứng với các thuộc tính được đặc trưng bởi kiểu dữ liệu và mã thuộc tính. Giá trị thuộc tính có kiểu dữ liệu được định nghĩa bởi thuộc tính.

Bằng cách tổ chức như vậy, hệ thống có thể thêm các bộ thuộc tính mới dễ dàng, cũng như bổ sung các thuộc tính vào các bộ thuộc tính đã có.

Ngoài ra một điểm khác cần lưu ý khi thiết kế mô hình sản phẩm đó là: một sản phẩm có nhiều hình ảnh, có thể thuộc nhiều danh mục (category) cũng như có thể có nhiều biến thể khác nhau (có quan hệ liên hệ với nhiều product khác). Ví dụ: điện thoại iphone có nhiều dòng sản phẩm

có màu sắc khác nhau; hay một chiếc váy có nhiều kích cỡ khác nhau.

2- THIẾT KẾ TẦNG NGHIỆP VỤ (BUSINESS LAYER)

Các nghiệp vụ sẽ được phát triển xoay quanh mô hình đã được xây dựng. Đây là tầng (layer) phức tạp nhất, chiếm phần lớn khối lượng công việc. Nghiệp vụ quản lý sản phẩm tập trung vào việc thay đổi các thuộc tính đa dạng của sản phẩm.

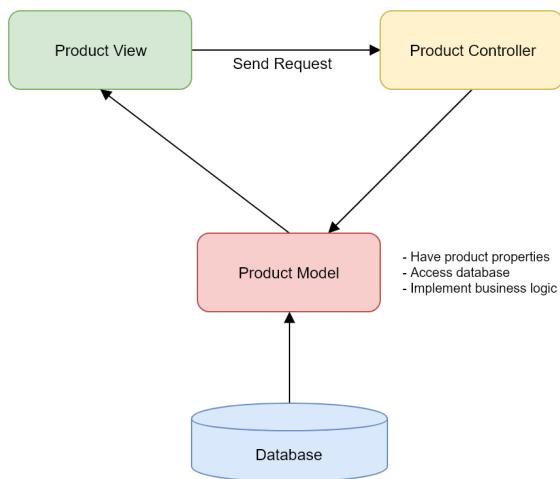
Khi thiết kế tầng này, người kỹ sư cần đảm bảo các mục tiêu thiết kế như:

- Đảm bảo nguyên lý **không lặp lại** (Don't Repeat Yourself) một cách tốt nhất, giúp các nghiệp vụ được phân tách rõ ràng, cô đọng và tập trung.
- Có thể mở rộng, sửa đổi các thành phần một cách độc lập.
- Linh hoạt trong việc phát triển.
- Có khả năng dễ dàng kiểm tra và bảo trì

Các mục tiêu đó sẽ ảnh hưởng tới các cách thiết kế mô hình ứng dụng khác nhau. Phần dưới đây sẽ phân tích từng mô hình thiết kế phần mềm để đưa ra sự lựa chọn phù hợp nhất.

2.1- Mô hình Model-View-Controller (MVC)

Mô hình MVC là mô hình phổ biến thường được lựa chọn để xây dựng ứng dụng. Đó là mô hình đơn giản, dễ tiếp cận và giúp các kỹ sư nhanh chóng đưa ra được các tính năng mới. Theo mô hình MVC, thì phần quản lý danh mục (catalogue) sẽ có kiến trúc như sau:



Với mô hình MVC, mô hình sản phẩm (product model) sẽ chứa tất cả các định nghĩa thuộc tính của sản phẩm, thông tin truy xuất cơ sở dữ liệu (database), cách thực thi các nghiệp vụ. Thông thường mô hình MVC sẽ sử dụng mẫu thiết kế (pattern) Active Record để ánh xạ (map) với cơ sở dữ liệu.

Đây là cách làm đơn giản nhưng cũng có các hạn chế sau:

- Mô hình sản phẩm trở nên quá nặng nề khi chứa quá nhiều logic về định nghĩa thuộc tính, truy xuất dữ liệu, thực thi nghiệp vụ...
- Do không phân tách được các tầng nghiệp vụ và dữ liệu nên khó thực hiện kiểm tra từng đơn vị riêng biệt (unit test).
- Việc thay đổi logic truy xuất dữ liệu và logic nghiệp vụ khó khăn do các thành phần bị phụ thuộc vào nhau.
- Tính đóng gói của nghiệp vụ không đảm bảo. Các nghiệp vụ sẽ chỉ được cài đặt theo mô hình CRUD. Vì vậy càng về sau sự trùng lặp logic càng xảy ra nhiều.

Do các hạn chế nêu trên nên đây không phải là mô hình phù hợp với các ứng dụng logic phức tạp như eCommerce do sẽ dẫn tới chi phí bảo trì cao cho hệ thống sau này..

2.2. Mô hình ba lớp (domain driven design)

Hướng tiếp cận tiếp theo là phân tích và thiết kế theo phương pháp Domain Driven Design.

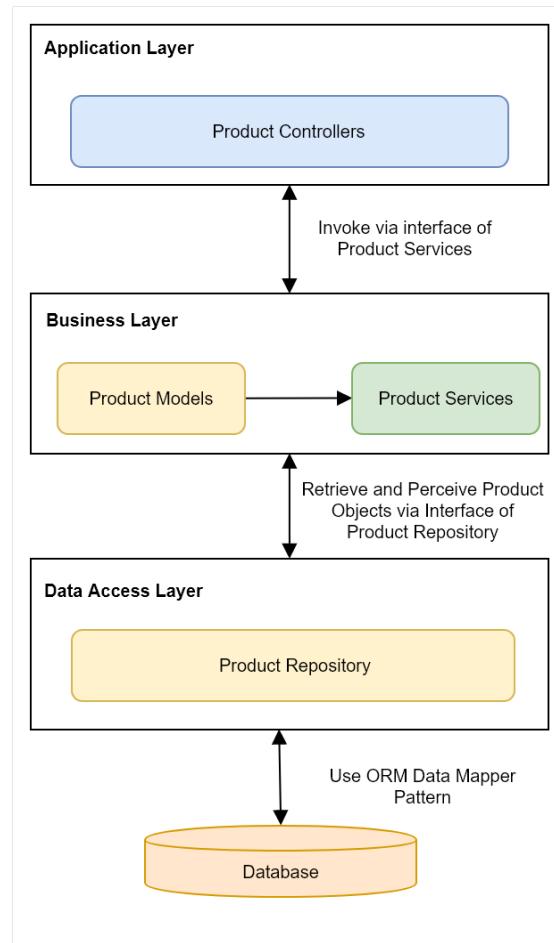
Đặc trưng của mô hình này là:

- Phân tách tầng nghiệp vụ (domain layer) ra khỏi tầng ứng dụng (application layer) và tầng truy xuất cơ sở dữ liệu (data access layer). Tầng nghiệp vụ sẽ trở thành nơi trung tâm chứa tất cả các logic nghiệp vụ.
- Trong tầng nghiệp vụ, cần tập trung vào thiết kế mô hình sao cho mô hình phản ánh đầy đủ nhất tính chất nhất quán của nghiệp vụ.
- Chia tầng nghiệp vụ thành hai thành phần riêng biệt: Domain Model và Domain Service. Domain service có

vai trò cung cấp các nghiệp vụ ra bên ngoài xoay quanh các domain model của hệ thống.

- Thiết kế riêng tầng hạ tầng (infrastructure) chứa các logic về truy xuất dữ liệu, thao tác với cơ sở dữ liệu, hàng đợi thông điệp (message queue).

Với việc tổ chức như thế này, các tầng ứng dụng, tầng



nghiệp vụ và tầng truy xuất dữ liệu sẽ được chia tách riêng biệt và chỉ tương tác với nhau thông qua giao diện. Các nghiệp vụ sẽ xoay quanh mô hình sản phẩm. Tầng truy xuất dữ liệu sẽ truy xuất hoặc lưu trữ các đối tượng thông tin về sản phẩm.

Nhưng việc áp dụng mô hình ba lớp xoay quanh mô hình sản phẩm cũng gây ra trở ngại cho việc xây dựng mô hình cho các nghiệp vụ đọc dữ liệu.

Các nghiệp vụ này rất đa dạng và cách làm hiệu quả nhất là sử dụng các câu truy vấn SQL trực tiếp. Nhưng nếu việc truy xuất bị thắt chặt vào các mô hình ORM (Object-Relational Mapping) sẽ làm ảnh hưởng tới hiệu năng, cũng như gây khó cho việc thay đổi các logic truy xuất dữ liệu. Đồng thời nó cũng ảnh hưởng tới tốc độ phát triển khi việc cài đặt các nghiệp vụ truy vấn đa dạng phục vụ phía ứng dụng giao diện bị cản trở và phụ thuộc vào việc thiết kế mô hình và tầng Repository.

MỘT HỆ THỐNG QLDMSP TỐT PHẢI ĐÁP ỨNG CÁC MỤC TIÊU SAU

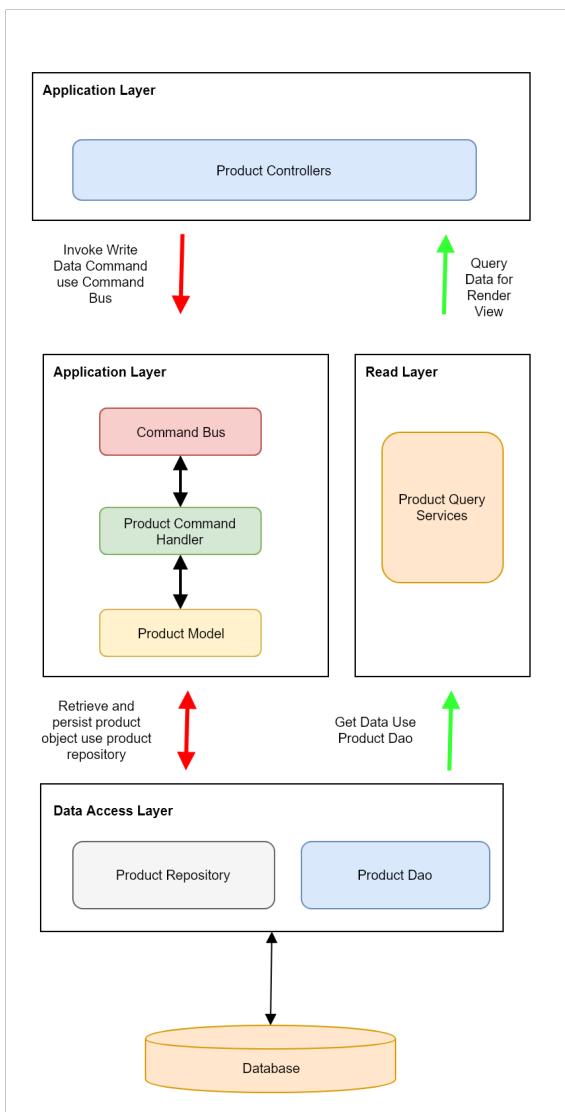
Quản lý được sự đa dạng về
thuộc tính của sản phẩm **1** Dễ dàng bảo trì và mở rộng về nghiệp vụ

Dễ dàng bảo trì và mở rộng về nghiệp vụ **3** **4** Có thể đáp ứng được yêu cầu cao về hiệu năng

2.3. Mô hình CQRS

Mô hình CQRS (Command Query Responsibility Segregation) là sự mở rộng của mô hình ba lớp trong DDD.

Đặc trưng quan trọng của CQRS là việc tách hai phần logic đọc và ghi dữ liệu ra thành hai phần riêng biệt độc lập với nhau:



- Phần ghi dữ liệu: được thực hiện qua việc gửi các lệnh (command) tới các hàm xử lý (handler) thông qua các tuyến lệnh (command bus). Bộ xử lý lệnh (Command handler) đóng vai trò tương tự domain service sẽ tương tác với các mô hình để thực hiện các nghiệp vụ thay đổi dữ liệu.
- Phần đọc dữ liệu: được thiết kế riêng không lệ thuộc vào các mô hình của phần ghi dữ liệu. Do đó có thể linh hoạt trong việc truy xuất cơ sở dữ liệu cũng như sử dụng các nguồn dữ liệu (data source) khác nhau để tối ưu về tốc độ truy xuất.

Mô hình CQRS đã khắc phục các hạn chế đã nêu ở mục 2 bên trên.

CQRS sẽ giúp mang lại các lợi thế lớn:

- Cho phép phát triển và tối ưu phần đọc dữ liệu tách biệt với phần ghi dữ liệu
- Việc mô hình hóa các nghiệp vụ ghi dữ liệu dưới các lệnh (command) cho phép che đậy tốt các logic nghiệp vụ, giúp việc mở rộng hệ thống dễ dàng hơn. Đồng thời các lệnh đó cũng có thể dễ dàng chuyển đổi giữa xử lý đồng bộ và bất đồng bộ thông qua lớp trung gian là tuyến lệnh (command bus) mà không cần phải thay đổi mô hình. Điều này sẽ giúp cung cấp một mô hình nhất quán, xuyên suốt trong bộ kiến trúc.

Qua phân tích trên, ta có thể nhận thấy cả ba mô hình trên đều có ưu nhược điểm riêng:

- Mô hình MVC đơn giản, nhưng có nhiều hạn chế khi giải quyết nghiệp vụ phức tạp cũng như gây trở ngại cho công đoạn kiểm tra (test) và mở rộng hệ thống sau này.
- Mô hình ba lớp DDD phù hợp cho việc xử lý nghiệp vụ phức tạp nhưng có nhiều hạn chế khi tối ưu phần đọc dữ liệu.
- Mô hình CQRS mở rộng từ mô hình ba lớp, giải quyết tốt việc chia tách đọc ghi, nhưng cũng đòi hỏi phải thiết kế phức tạp hơn.

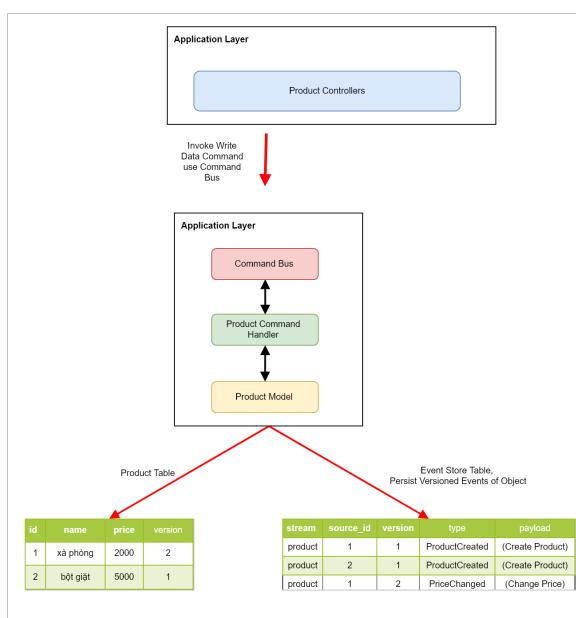
Đối với các hệ thống eCommerce lớn thì mô hình CQRS là mô hình phù hợp nhất vì nó giúp giải quyết cả hai đòi hỏi lớn là xử lý nghiệp vụ phức tạp và đáp ứng hiệu năng cao. Phân tiếp theo sẽ phân tích sâu hơn về mặt hệ thống và một biến thể mở rộng của CQRS là CQRS - ES (Command Query Responsibility Segregation - Event Sourcing).

3. THIẾT KẾ VỀ HỆ THỐNG

Một trong các đòi hỏi lớn của việc quản lý danh mục sản phẩm là phải thiết kế để đáp ứng các nhu cầu về hiệu năng và tích hợp với các hệ thống khác. Để đáp ứng các yêu cầu tốt, thì phải được thiết kế từ tổng thể ngay từ đầu. Có vậy mới tạo ra sự phát triển liên mạch, nhất quán, giúp đảm bảo tính ổn định cũng như tiến độ làm việc.

3.1. Mô hình CQRS - ES

Mô hình Event Sourcing - ES là mô hình thiết kế mà trạng thái của đối tượng (object) sẽ được lưu trữ dưới dạng chuỗi các sự kiện thay đổi. Nó khác với mô hình thiết kế thông thường vốn chỉ lưu trữ trạng thái cuối cùng của đối tượng.



Khi đối tượng bị thay đổi, hệ thống sẽ tạo ra các phiên bản sự kiện (versioned event) để lưu trữ dữ liệu thay đổi của đối tượng. Các sự kiện (event) sẽ được lưu trữ bằng cách chèn thêm mới (Appending Only) vào một cấu trúc bảng gọi là kho lưu trữ sự kiện (event store). Việc lưu trữ thông tin dưới dạng các phiên bản sự kiện này mang lại các lợi ích:

- Lưu trữ được lịch sử thay đổi của các đối tượng.
- Nếu các sự kiện được gửi tới các hệ thống khác, các hệ thống đó có thể tự tái hiện lại trạng thái cuối cùng chính xác của đối tượng gốc, do đó dễ dàng tích hợp với các hệ thống khác.
- Dựa trên các sự kiện đã được tạo ra, có thể xây dựng phần lưu trữ riêng cho khâu đọc (read side), để tối ưu hóa tốc độ truy xuất dữ liệu.

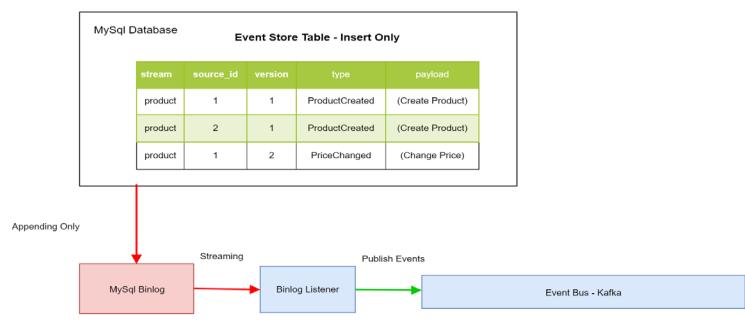
3.2. Event Bus - Kafka - MySQL Binlog

Các phiên bản sự kiện (versioned event) của đối tượng nếu được gửi đi sẽ giúp hệ thống dễ dàng tích hợp với nhiều hệ thống khác. Event Bus là tên gọi logic của một đường truyền chứa tất cả các phiên bản sự kiện dưới dạng luồng

(stream). Các thành phần cần xử lý dữ liệu (consumer) chỉ cần đăng ký nhận sự kiện từ ống dữ liệu (subscribe event bus) là có thể nhận được các sự kiện để xử lý.

Các sự kiện được lưu trữ trong cơ sở dữ liệu mysql. Tất cả các thay đổi trong cơ sở dữ liệu đều được MySQL lưu vào một log file, gọi là binlog. Bằng việc sử dụng binlog của mysql, thì hệ thống có thể bắt được mọi event phát sinh để gửi đi.

Kafka là một bộ phận điêu phối thông điệp (message broker) theo mô hình lưu trữ kiểu log (log-based) cho phép lưu trữ và gửi đi các sự kiện với hiệu suất cao nhưng vẫn đảm bảo đúng với thứ tự gửi vào. Vì vậy nó là lựa chọn tốt nhất để làm tầng lưu trữ cho event bus.



3.3. Kiến trúc tổng thể

Đến đây bức tranh của hệ thống đã trở nên rõ ràng. Từng mảnh ghép từ mô hình, nghiệp vụ tới tích hợp đã đầy đủ. Có thể tóm tắt lại: hệ thống sẽ xây dựng nghiệp vụ quanh mô hình sản phẩm; cấu trúc theo mô hình CQRS - ES; sử dụng MySQL Binlog; Kafka để publish các sự kiện thay đổi một cách ổn định; dựa trên event bus sẽ xây dựng các cấu trúc lưu trữ có tốc độ truy xuất cao như Elastic, MongoDb, cũng như tích hợp với các hệ thống khác.

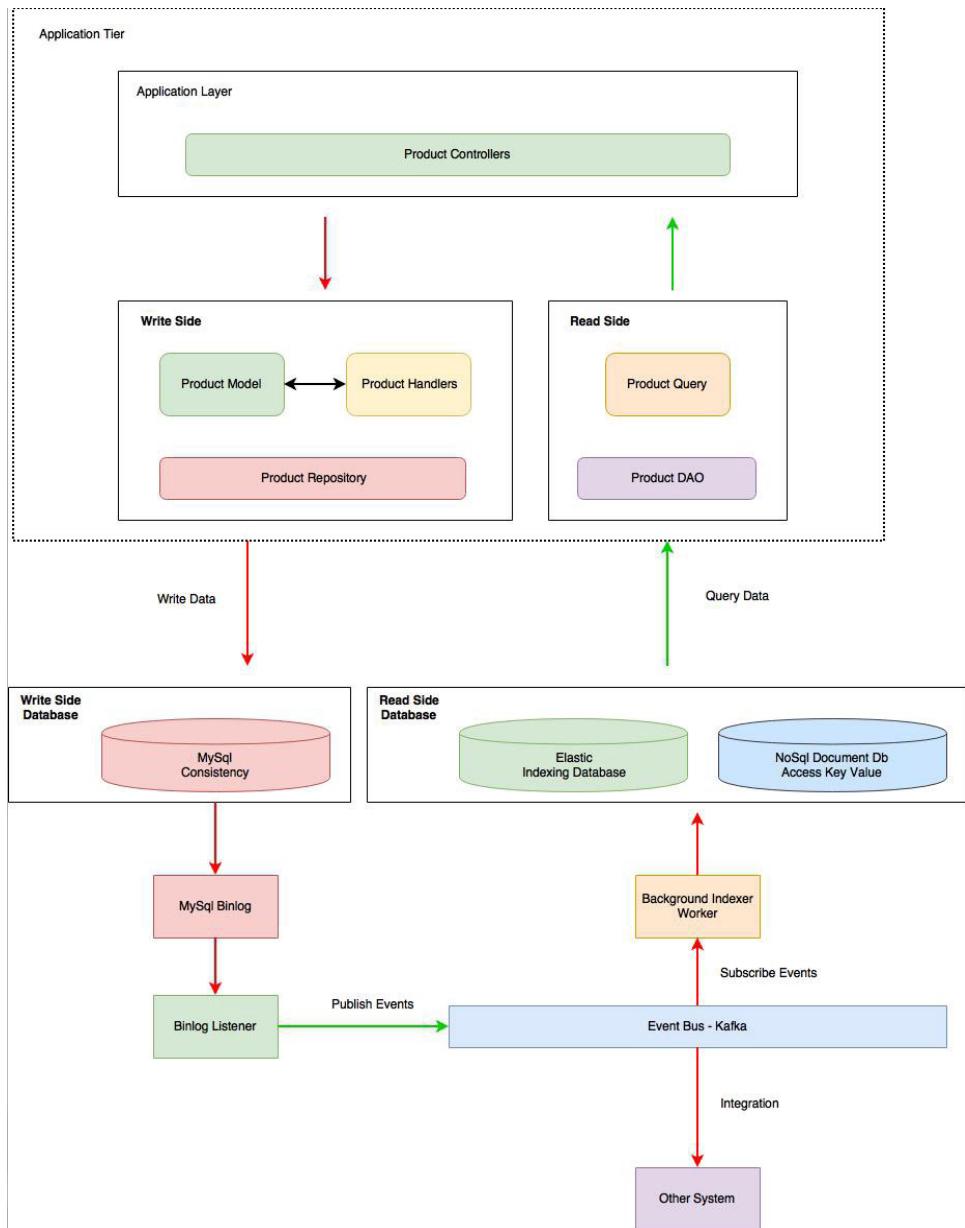
4. KẾT LUẬN

Thiết kế hệ thống quản lý danh mục sản phẩm trong eCommerce đòi hỏi phải đạt được cùng lúc nhiều mục tiêu, với một tầm nhìn xuyên suốt và nhất quán. Từ mức ứng dụng cho tới mức hệ thống đều phải có sự gắn kết liên mạch. Tất cả hướng tới hai mục tiêu quan trọng phải đạt được:

- Xử lý được các nghiệp vụ phức tạp
- Đảm bảo hiệu năng và độ ổn định của hệ thống.

Phương pháp phân tích thiết kế Domain Driven Design, mô hình CQRS - ES, cơ chế replicate ổn định của MySQL, sự đảm bảo thứ tự message của Kafka, cũng như các database được thiết kế tối ưu cho thao tác đọc như Elastic, Mongo... là các nhân tố chính đảm bảo chất lượng của hệ thống.

SƠ ĐỒ KIẾN TRÚC TỔNG THỂ



Đọc và thảo luận bài viết online:
<https://engineering.grokking.org/>



Nguồn: Internet

Lock free programming

● Vấn đề với lock

Là một lập trình viên từng làm việc với phần mềm đa nhiệm (multi-thread), khái niệm lock chắc chắn không hề xa lạ với bạn. Khi nhiều luồng (thread) cùng làm việc với một cấu trúc dữ liệu chung (shared data structure) thì chúng ta cần có một cơ chế để giới hạn quyền truy cập, chỉ cho phép một luồng (thread) duy nhất truy cập và sửa đổi dữ liệu đó. Cơ chế phổ biến nhất cho việc này chính là lock.

Tranh chấp lock (Lock contention) chính là điểm mấu chốt dẫn đến hiệu năng không được như mong muốn khi lập trình song song.

Mặc dù lock là một kĩ thuật rất phổ biến và không tránh khỏi trong hầu hết các bài toán lập trình nhưng làm việc tốt với các kĩ thuật lock là không hề dễ dàng và dễ dẫn đến các sai lầm, trong đó sai lầm phổ biến nhất là các lập trình viên có xu hướng lock quá nhiều một cách không cần thiết

dẫn đến giảm hiệu năng của ứng dụng, cũng như mắc phải các sai lầm về logic dẫn đến deadlock hay là luồng bị chết đói (starvation) do không có tài nguyên về CPU.

Dưới đây chúng ta hãy thử duyệt qua một vài điểm yếu nữa của việc sử dụng lock:

Dead lock là chuyện xảy ra khi 2 ứng dụng cùng đợi tài nguyên lẫn nhau dẫn đến gây ra vòng lặp vô tận.

Priority inversion là khi một tiến trình (process) có độ ưu tiên thấp lại lock một tài nguyên thuộc về một tiến trình có độ ưu tiên cao hơn.

Convoying problem nói về vấn đề gặp phải khi một loạt tiến trình phụ thuộc chung vào một tài nguyên nhưng tiến trình chạy chậm nhất lại đang giữ lock đầu tiên, khiến cho những tiến trình chạy nhanh hơn phải chờ.

Async-signal safety vấn đề này thường xảy ra trong lập trình hệ thống khi một hàm hệ thống (system call) có thể bị ở trạng thái gián đoạn (interrupt) bởi một tín hiệu

Về Kipalog

Kipalog (viết tắt của keep-a-log) là một dự án được tạo nên với quan niệm: kiến thức mà không chia sẻ là kiến thức chết. Tại đây bạn có thể viết bất kì kiến thức kĩ thuật nào bạn muốn giữ lại, miễn là kiến thức đó có ích với ai đó, kể cả chỉ một mình bạn. Từ lúc thành lập Kipalog đã thu hút được hàng ngàn bài chia sẻ kiến thức hữu ích. Kipalog team hy vọng đây sẽ trở thành nơi mà bất kì ai yêu thích kĩ thuật cũng sẽ ghé qua, và để lại chút kiến thức có ích cho người khác thông qua việc viết bài cũng như bình luận.

(signal) nào đó như sigkill hay sighup.

Vậy điều gì xảy ra khi bạn có một hàm xử lý (handler) cho tín hiệu sigkill mà khi đang thực thi hàm đó thì một tín hiệu sigkill khác lại được gửi đến. Bạn sẽ nghĩ ngay đến việc khóa lại việc xử lý của hàm xử lý này, tuy nhiên việc đó sẽ gây ra deadlock tức thời (immediate deadlock) bởi bản thân hàm xử lý sẽ cố gắng lấy cái lock mà chính bản thân nó đang giữ.

Một thao tác được gọi là Async-Signal-Safe khi nó đảm bảo không làm phiền đến các tác vụ khác khi mà chúng đang bị gián đoạn, tuy nhiên vô cùng đáng buồn là hai tác vụ malloc và free lại không nằm trong danh sách các thao tác thuộc nhóm Async-Signal-Safe này.

● Về lock free programming (hay lockless programming)

Như cái tên của nó, lock-free (hay lockless) là các kĩ thuật lập trình song song nhằm đảm bảo tài nguyên có thể được chia sẻ hoặc thay đổi một cách an toàn mà không cần khởi tạo hay giải phóng lock, cũng như không cần sử dụng bất kì kĩ thuật lock khác như semaphore hay mutex.

Là một kĩ thuật hướng đến sự hoàn hảo trong lập trình đa luồng (multi thread programming), lock-free là một kĩ thuật rất khó và không thể thực hiện được nếu thiếu sự hỗ trợ của phần cứng.

Một trong những bài báo lâu đời và nổi tiếng về kĩ thuật lập trình lock-free là của Leslie Lamport (Turing award

holder) những năm 1977 với tựa đề là “Concurrent Reading and Writing” [1]. Trong bài báo này, tác giả đề cập đến một cấu trúc dữ liệu cho lock-free gọi là non-blocking buffer cũng như giải thích thêm những giới hạn của cấu trúc đó.

Như đã nói ở trên, lock free cần sự hỗ trợ của phần cứng, thông qua các phép toán cơ bản (primitive) đơn giản mà phần cứng hỗ trợ. Các phép toán này còn gọi là **ATOMIC OPERATION** (hay gọi là các thao tác “nguyên tử”, tức là không thể chia nhỏ hơn được nữa). Điểm đặc biệt của các thao tác này đó là khiến cho một luồng không thể quan sát được một hành động diễn ra giữa chừng (half-complete). Một số ví dụ về các phép toán đó như là CAS (Compare and set), LL/CS (Load linked, store condition), MCAS (Multi-word CAS), CAS2, CAS/N.

COMPARE AND SET (CAS) là phép toán mà trước khi thay đổi một biến số nào đó thì CPU sẽ phải so sánh nó với giá trị trước khi hàm CAS được gọi, nếu giá trị khác nhau (do đã bị thay đổi bởi một luồng khác) thì sẽ không cho phép thực hiện phép thay đổi đó.

Ngoài ra thay vì thiết kế một giải thuật tổng quát, thì thường các nhà thiết kế giải thuật sẽ thiết kế các cấu trúc dữ liệu lock free (*lock-free data structures*) để thay thế. Và các giải thuật chỉ cần dựa trên các cấu trúc dữ liệu này thì cũng đã giải quyết được khá khá các bài toán song song. Dưới đây chúng ta sẽ thử xem qua một vài cấu trúc dữ liệu:

1. LOCK-FREE STACK (LIFO QUEUE)

```
class Node {  
    Node * next;  
    int data;  
};  
Node * head;
```

• Thao tác Push

```
void push(int t) {  
    Node* node = new Node(t);  
    do {  
        node->next = head;  
    } while (!cas(&head, node, node->next));  
}
```

- **Thao tác Pop**

```
bool pop(int& t) {
    Node* current=head;
    while(current) {
        if(cas(&head, current->next, current)) {
            t = current->data;
            return true;
        }
        current = head;
    }
    return false;
}
```

Ở đây câu lệnh cas(*Node x, *Node oldValue, *Node newValue) sẽ so sánh giá trị của biến X với oldValue (giá trị cũ). Nếu giá trị của biến X bằng với giá trị oldValue thì hàm cas sẽ được thực thi, gán giá trị newValue vào cho biến X. Đối với trường hợp giá trị của biến X không bằng với giá trị oldValue thì hàm cas sẽ không được thực thi (do đã có thread khác tác động lên biến X dẫn đến giá trị thay đổi).

Bạn có thể thấy một cấu trúc dữ liệu stack dạng lock free chỉ đơn thuần là một stack có áp dụng các phép toán atomic kết hợp với vòng lặp while.

2. DANH SÁCH LIÊN KẾT DẠNG LOCK-FREE

Danh sách liên kết dạng lock-free (lock-free linked list) là một ví dụ rất tốt để chứng minh vấn đề hiệu năng giữa việc sử dụng lock hay không sử dụng lock.

Với danh sách liên kết (linked list) thì đôi khi bạn sẽ phải duyệt qua một quãng đường dài để tìm kiếm một phần tử nào đó. Vì vậy nếu sử dụng lock thông thường thì bạn sẽ phải lock toàn bộ danh sách khi cần duyệt!! Cho nên việc vận dụng cấu trúc dữ liệu Lock-free trong tình huống này sẽ giúp tiết kiệm thời gian rất nhiều.

Cách hiện thực danh sách liên kết dạng lock-free khá phức tạp do có liên quan đến thao tác xóa nên các bạn có thể tham khảo thêm chi tiết trong bài viết “Generic Concurrent Lock-Free Linked List” [2].

Ngoài ra có một bài báo khác khá nổi tiếng về danh sách liên kết dạng lock-free với tiêu đề “A Pragmatic Implementation of Non-Blocking Linked-Lists” [3] mà bạn đọc có thể tham khảo thêm.

● Bài toán ABA

Với lock free sử dụng atomic operation sẽ xảy ra một hiện tượng rất thú vị gọi là ABA, hiện tượng đó được diễn giải như dưới đây:

- Thread 1 nhìn vào một biến chia sẻ X và định thay đổi nó. Lúc thread 1 nhìn thì X có giá trị là A.
- Thread 1 tính toán một vài thứ thú vị **dựa trên sự thật là X có giá trị là A**
- Thread 2 chạy, biến đổi X thành giá trị là B.

- Thread 1 gọi CompareAndSet nhìn thấy giá trị X đã bị đổi, sau đó thử lại.
- Thread 2 thay đổi lại giá trị X là A.
- Thread 1 thực hiện lại và thành công với CompareAndSet.

Vấn đề ở đây chính là việc Thread 1 “tưởng” là mình đã gán thành công khi biến X không bị thay đổi từ bên ngoài nhưng thực ra X đã bị thay đổi mà Thread 1 không hề biết. Khi stack dạng lock-free gặp phải vấn đề này thì sẽ có khả năng bạn đã gọi thao tác pop 2 lần trên cùng một node và sẽ bị tình trạng segfault.



Để vượt qua vấn đề này thì chúng ta có thể thêm một biến là “update count” dựa vào khái niệm “doubleword CAS” (cần sự hỗ trợ phân cứng).

● Chú ý và kết luận

Lock free không phải là thuốc chữa cho bài toán tranh chấp tài nguyên. Việc nhiều thread cùng tranh chấp một tài nguyên vẫn có thể xảy ra, chỉ khác là ở tần suất thấp hơn, và được giải quyết tốt hơn (về mặt giải thuật + sự hỗ trợ về phần cứng). Starvation (chết đói), hay là hiện tượng khi một vài thread không nhận được resource đủ để chạy vẫn có thể xảy ra.

Để áp dụng lock free thì bạn cần lưu ý hai yếu tố:

- Bài toán đủ đơn giản.
- Bạn tự nghĩ ra được một thuật toán lock free để áp dụng cho một vài trường hợp hợp.

Lock free sẽ làm tăng hiệu năng cho ứng dụng nhưng để hiện thực đúng là hoàn toàn không dễ (vấn đề ABA là một ví dụ).



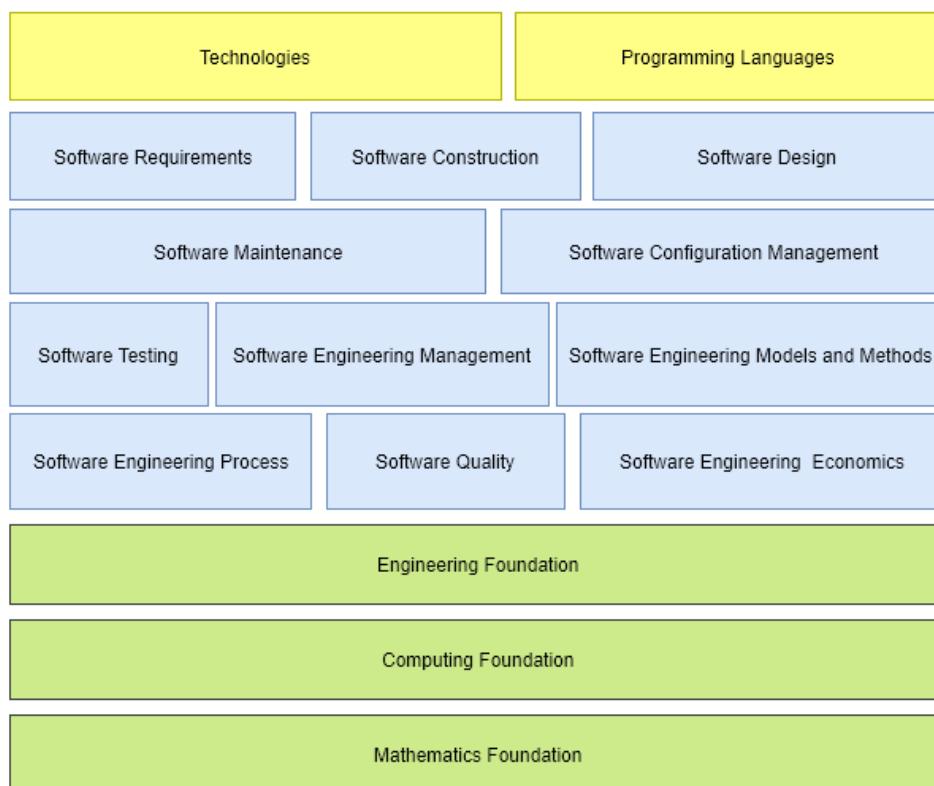
Đọc và thảo luận bài viết online:
<https://engineering.grokking.org/>

Đây là một sơ đồ bao gồm các mảng kiến thức được tổng hợp bởi IEEE Computer Society, một tổ chức quy tụ những kỹ sư giàu kinh nghiệm làm việc trong ngành phần mềm, nhằm cung cấp một góc nhìn tổng quan cho những người làm phần mềm chuyên nghiệp về những mảng kiến thức mà một người cần nắm.

Bạn đã biết được bao nhiêu mảng kiến thức được mô tả dưới đây?



Software Engineering Knowledge Areas



& HTTP/1 HTTP/2

LÊ NGUYỄN DŨNG | Thuộc lứa 8x cuối cùng, hiện là kỹ sư backend cho một ứng dụng OTT quen thuộc với người dùng Việt Nam, đang ẩn cư tại Sài Gòn.

Nguồn: Internet



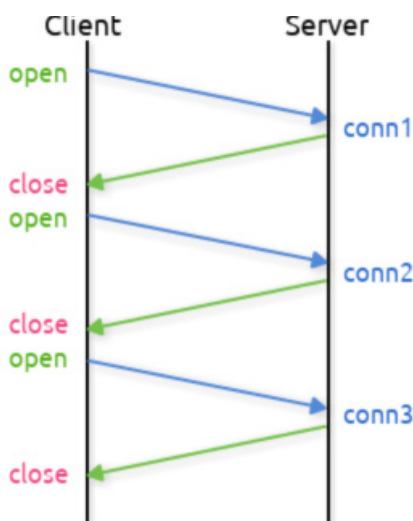
Năm 1989, Tim Berners-Lee phát minh ra HTTP. Năm 1996, HTTP/1.0 được tổ chức RFC thông qua trở thành một chuẩn (standard), HTTP/1.1 cũng hình thành cùng năm đó, đây là một trong những cột mốc quan trọng bậc nhất khiến thế giới công nghệ phát triển như vũ bão và định hình như ngày hôm nay. Sau 26 năm kể từ ngày phát minh và gần 20 năm kể từ ngày phiên bản 1.0 xuất hiện, tháng 5 năm 2015, tổ chức RFC đã thông qua đặc tả HTTP/2, đánh dấu thay đổi lớn nhất kể từ ngày hình thành tới nay của giao thức HTTP.

Nội dung của bài viết này sẽ không đi sâu vào chi tiết đặc tả của HTTP/2, thay vào đó người viết sẽ cố gắng nhìn nhận vấn đề cốt lõi của việc nâng cấp này thông qua tập trung phân tích hai cơ chế: **kết nối** và **truyền dữ liệu**.

HTTP/1

Đầu tiên, chúng ta quan sát lại cách làm của HTTP/1:

1. KẾT NỐI CÓ VÒNG ĐỜI NGẮN



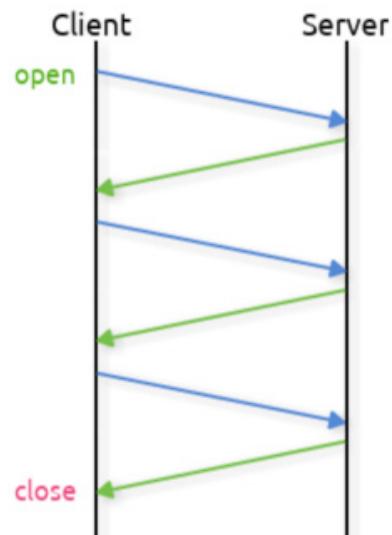
Hình: HTTP/1 connection có vòng đời ngắn

Trong dạng xử lý này, mỗi kết nối (connection) sẽ yêu cầu một tài nguyên độc lập. Kết nối sẽ được tạo mới khi cần yêu cầu một tài nguyên và đóng lại sau khi nhận được dữ liệu trả về. Đây là dạng làm việc đơn giản và cũng là

thông dụng nhất.

Tuy nhiên cách làm này rất không tối ưu vì thời gian thiết lập kết nối mới tốn rất nhiều thời gian do phải thực hiện quá trình bắt tay 3 bước (three-hand-shaking), chưa kể nếu có các lớp xử lý bảo mật như TLS sẽ càng làm tăng độ trễ của việc thiết lập kết nối này.

2. KẾT NỐI CÓ VÒNG ĐỜI DÀI HẠN (KEEP-ALIVE CONNECTION)

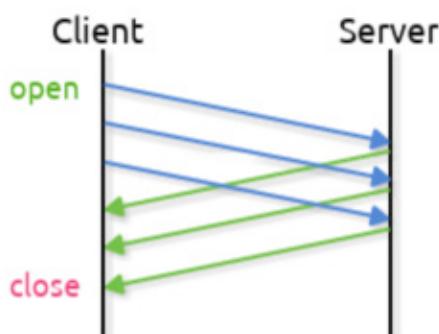


Hình: Connection keep-alive

Với keep-alive connection, có thể thực hiện nhiều lượt yêu cầu tài nguyên khác nhau với cùng một kết nối.

Tuy nhiên, có thể nhận thấy một khoảng thời gian chờ sau khi thực hiện một request, trong lúc đợi response về hoàn tất, sẽ không có một request nào khác được phép thực hiện. Điều này gây lãng phí tài nguyên, do băng thông vẫn còn đó trong khi request tài nguyên khác sẽ không thể request ở cùng kết nối (connection), và nếu tạo ra kết nối mới thì lại trở lại vấn đề với mỗi request một kết nối.

3. HTTP/1 PIPELINING



Hình: HTTP pipelining

Giải pháp cho phép một loạt request được gọi mà không cần response phải hoàn thành, sau đó sẽ nhận các response. Một giải pháp khá mượt, nhưng nhìn sâu hơn vấn đề một chút, đầu tiên ta quan sát lại HTTP request và response.

Một cú pháp request điển hình như sau:

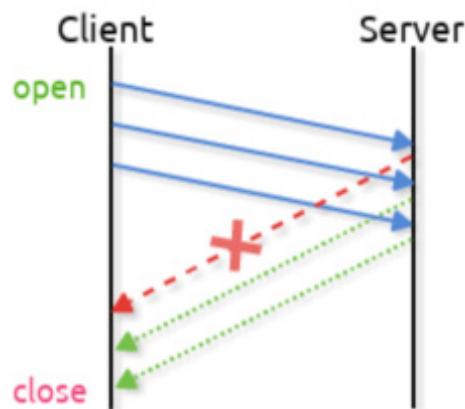
GET /index.html HTTP/1.1
Host: www.example.com

Cú pháp response dạng:

HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
ETag: "3f80f-1b6-3e1cb03b"
Content-Type: text/html; charset=UTF-8
Content-Length: 138
Accept-Ranges: bytes
Connection: close
<html>
<head>
<title>An Example Page</title>
</head>
<body>
Hello World, this is a very simple HTML document.
</body>
</html>

Nếu dữ liệu trả về không có thứ tự mà đi lộn xộn, thì do cùng sử dụng 1 connection chung, đầu nhận response sẽ không thể phân biệt được phần dữ liệu là thuộc request nào.

Điều này khiến HTTP pipelining phải tuân thủ một điều là phía client chỉ sau khi nhận trọn vẹn một response của request này mới được nhận tiếp response từ request khác, việc này gây ra nguy cơ về hiệu năng khi có một response bị mất hay bị xử lý chậm.



Hình: Head-of-line blocking trong HTTP pipelining

Vấn đề này được gọi là head-of-line blocking. Nếu vì một lý do nào đó request tới trước được phía server xử lý chậm hơn request sau thì các request sau nó cũng không được trả về ngay mà buộc phải đợi.

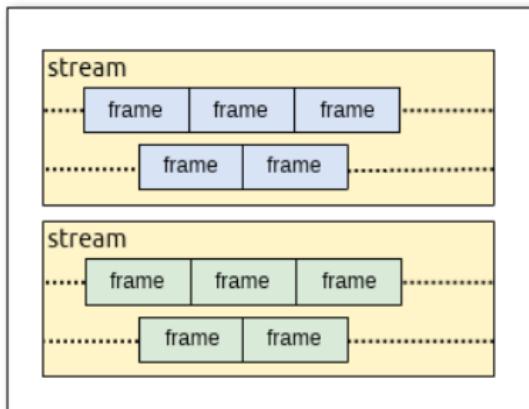
Vì vấn đề này mà hầu hết các trình duyệt dù có hỗ trợ sẵn HTTP pipelining cũng đều mặc định ở trạng thái tắt (disabled). Ở góc nhìn web server cũng không khá hơn mấy, một số web server sẽ gặp vấn đề khi xử lý kết nối dưới dạng HTTP pipelining.

Những vấn đề với HTTP pipelining khiến việc cải tiến tốc độ ở góc nhìn kết nối và truyền tải bị dừng lại. HTTP/1 cứ vậy vận hành với phiên bản 1.1 cho tới khi giao thức SPDY xuất hiện, đây là nền tảng cho việc xây dựng HTTP/2.

Năm 2009, Google giới thiệu giao thức SPDY, về cơ bản đây là một giao thức ở tầng TCP sử dụng binary thay vì text như trên HTTP/1. Giao thức cũng hỗ trợ sẵn cơ chế về nén dữ liệu, các lớp bảo mật, chế độ ưu tiên ... để tăng tốc độ truyền tải dữ liệu web.

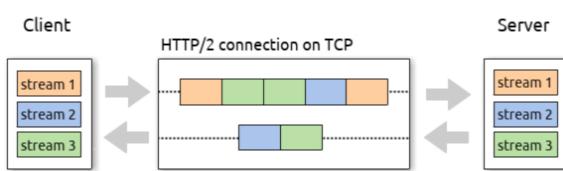
Năm 2015, trên nền tảng của SPDY, HTTP/2 được tổ chức RFC công nhận như một chuẩn. Tiến trình phổ cập HTTP/2 thay thế HTTP/1 đang diễn ra trên toàn cầu.

HTTP/2



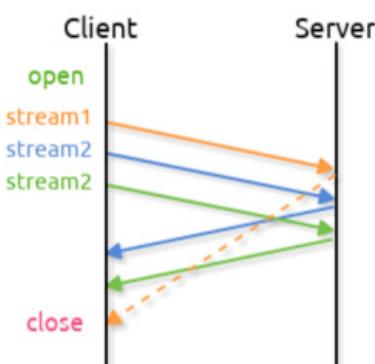
Hình: Kiến trúc HTTP/2

HTTP/2 được xây dựng trên tư tưởng mỗi TCP connection sẽ có nhiều stream, mỗi packet được truyền tải trong stream sẽ là một frame, các frame này sẽ có đầy đủ các thông tin định danh stream.



Hình: HTTP/2 nhiều stream dùng chung một connection

Khi sử dụng, mỗi request sẽ sử dụng một stream độc lập, mỗi frame đều có thông tin định danh cho stream riêng, nên việc thứ tự truyền/nhận của request/response không còn là vấn đề, dữ liệu của frame nào sẽ được đưa vào stream của nó, tương ứng với đó là request/response tương ứng. Như vậy vấn đề head-of-line blocking được giải quyết.



Hình: Vấn đề Head-of-line blocking được giải quyết trong HTTP/2

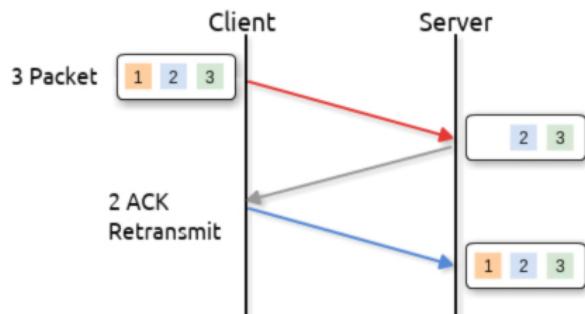
2. VĂN ĐỀ

a. Head-of-line blocking gây ra bởi TCP

Ý tưởng của HTTP/2 giúp tránh được HOL giữa các request, tuy nhiên lại mắc phải một dạng HOL khác gây ra bởi giao thức TCP.

HTTP/2 truyền nhiều request (stream) trên cùng một TCP connection. Nhưng TCP sẽ không biết về điều này. Dưới góc nhìn của TCP chỉ có những packet được đánh số thứ tự. Nhiệm vụ của TCP là đảm bảo tất cả packet được truyền đi đầy đủ, và đúng thứ tự. Do đó nếu có packet nào bị mất do truyền tải ở tầng IP thì tầng TCP sẽ huỷ toàn bộ các packet phía sau, và bắt đầu truyền lại từ packet bị mất.

Trong trường hợp những packet này thuộc những stream khác nhau, thì khi mất mát gói tin của một stream thì các stream còn lại sẽ bị ảnh hưởng. Và head-of-line blocking lại xuất hiện.



Hình: Head-of-line blocking do cơ chế bảo vệ flow của TCP

b. Stream dùng chung thuật toán chống nghẽn

Thuật toán chống nghẽn được thiết kế nhằm giúp TCP sử dụng băng thông tối đa mà nó có thể, tùy vào lượng mất mát gói tin mà thuật toán sẽ tự điều chỉnh lượng băng thông truyền đi.

Trong HTTP/2, nhiều stream sẽ dùng chung trên một kết nối, nên sẽ phải sử dụng chung một trạng thái chống nghẽn. Việc này sẽ gây ra vấn đề nếu một gói tin thuộc stream tương ứng bị mất, thì thuật toán chống nghẽn sẽ điều chỉnh làm chậm băng thông của kết nối lại, kéo theo các stream còn lại của kết nối này cũng sẽ bị chậm theo.

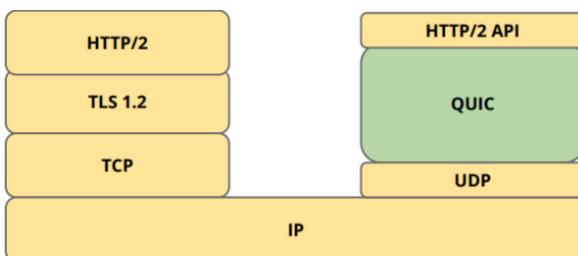
QUIC - QUICK UDP INTERNET CONNECTIONS

1. TỔNG QUÁT

Để khắc phục vấn đề head-of-line blocking gây ra bởi TCP, cách duy nhất là thiết kế HTTP/2 trên nền một giao thức khác. QUIC là 1 giao thức xây dựng trên nền UDP, một giao thức thông dụng và có thể dễ dàng tương tác ở userland.

Tư tưởng thiết kế giao thức QUIC là sự kết hợp giữa TCP và tinh thần của HTTP/2

- Đảm bảo không mất mát, đúng thứ tự gói tin: Tương tự TCP
- Cải thiện thuật toán chống nghẽn: Sử dụng thuật toán CUBIC
- Tự sửa lỗi gói tin
- Hỗ trợ lớp bảo mật TLS



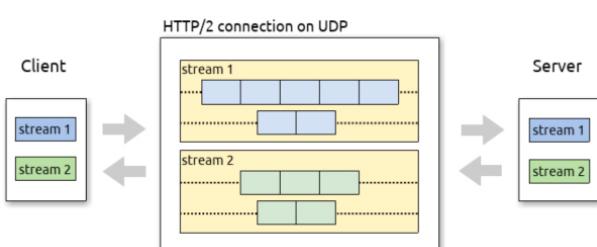
Hình: $QUIC = TLS + TCP + SPDY$

Mặc dù trong UDP, khái niệm kết nối không tồn tại, nhưng dựa trên tinh thần của TCP, QUIC xây dựng lại kết nối cho mình, tuy nhiên tầng truyền tải dữ liệu sẽ độc lập giữa các stream. Điều này giúp loại bỏ head-of-line blocking gây ra khi các stream truyền tải dùng chung kết nối. Một điều thú vị là kết nối của QUIC sẽ tốt hơn TCP trong một số trường hợp. Ví dụ khi kết nối mạng của bạn đang sử dụng đột ngột bị ngắt và kết nối lại, trong khi kết nối của TCP sẽ bị mất và buộc phải khởi tạo kết nối mới, thì QUIC do xây dựng trên nền tảng UDP - một giao thức phi kết nối (connectionless) nên về lý thuyết thì kết nối sẽ vẫn tiếp tục truyền nhận dữ liệu bình thường dù sẽ chậm đi một chút.

QUIC cũng sử dụng thuật toán chống nghẽn cho từng stream riêng biệt, loại bỏ vấn đề thuật toán chống nghẽn gây ra cho HTTP/2.

việc sử dụng rộng rãi giao thức này còn rất hạn chế. Hiện chỉ có hai trình duyệt hỗ trợ giao thức này là Chromium và Opera.

Việc cố gắng xây dựng lại một giao thức lõi của internet của Google là một diễn biến lạ lẫm, họ thử nghiệm QUIC trên những sản phẩm nội bộ của mình (các dịch vụ web và trình duyệt), nhưng Google đủ lớn để biến những cuộc thử nghiệm về công nghệ khiến cả thế giới công nghệ phải ôn ào và học hỏi, thậm chí trở thành chuẩn. Trường hợp đó đã đúng với Ajax, BigTable, SPDY ... và biết đâu đó là QUIC trong tương lai.



Đọc và thảo luận bài viết online:
<https://engineering.grokking.org/>

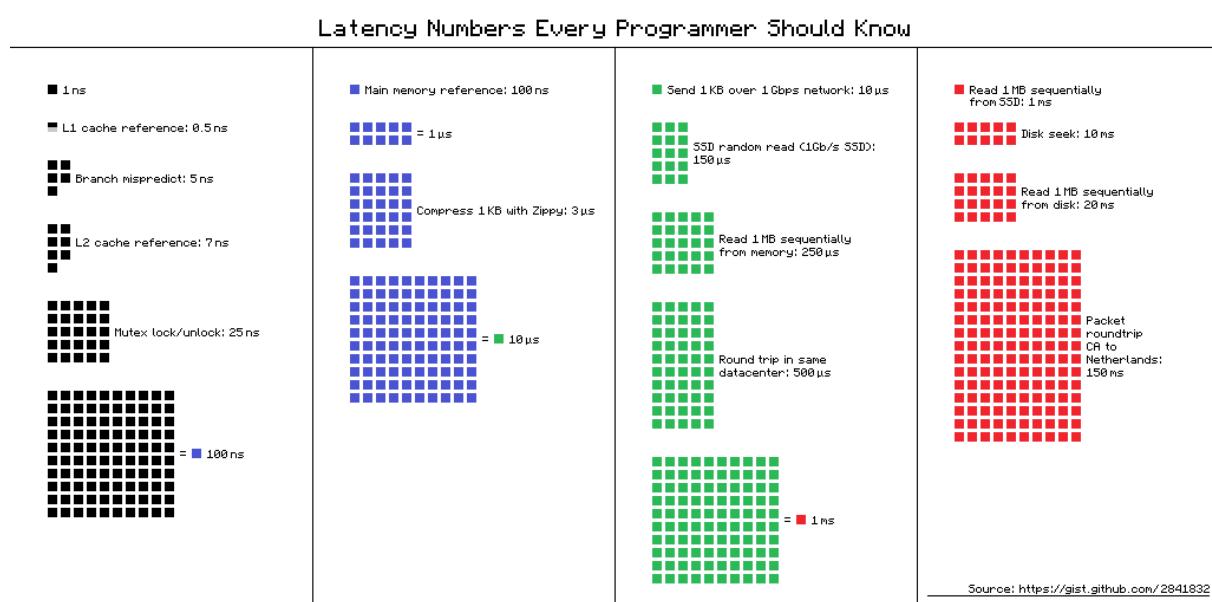
Hình: Giao thức QUIC

2. LỜI KẾT

QUIC hiện tại là một phần của dự án Chromium, chưa có một đóng gói độc lập nào để sử dụng giao thức này. Điều này khiến việc sử dụng QUIC rất khó khăn. Mặc dù rất nhiều các dịch vụ của Google hiện tại đã dùng QUIC nhưng



POSTGRESQL BUFFER hoạt động như thế nào?



GIỚI THIỆU

Hầu hết các hệ thống cơ sở dữ liệu SQL hiện nay đều lưu trữ dữ liệu ở trên ổ đĩa. Tuy nhiên, như chúng ta đã biết thì truy xuất từ đĩa sẽ chậm hơn khoảng 80 lần so với RAM. Dù bạn có thay thế đĩa cứng bằng ổ SSD thì vẫn sẽ chậm hơn 4 lần so với RAM.

Các hệ cơ sở dữ liệu SQL xử lý việc này bằng kỹ thuật buffer. Buffer hoạt động dựa trên nguyên tắc: đưa 1 phân thông tin thường dùng chứa thẳng trên RAM để có tốc độ truy xuất nhanh. Những lần sử dụng sau thì sẽ không phải truy xuất ổ đĩa để lấy dữ liệu nữa.

Trong PostgreSQL, hệ thống đảm nhận việc này có tên là Buffer Manager.

CẤU TRÚC CỦA BỘ QUẢN LÝ ĐỆM

Bộ quản lý bộ nhớ gồm có 3 thành phần: buffer table, buffer descriptors và buffer pool. Trong bài viết này chúng ta sẽ chỉ đề cập đến buffer pool.

Buffer pool chính là bộ nhớ tạm mà chúng ta đã nói qua ở kỹ thuật buffer. Buffer pool được hiện thực dưới dạng 1 mảng được lưu trữ trên RAM. Mỗi ô của mảng này sẽ chứa một lượng “dữ liệu thô”, và index của ô này được gọi là `buffer_id`.

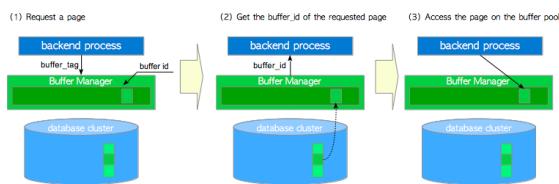
Buffer tag

Ai đã từng làm việc với SQL đều biết là dữ liệu của chúng ta được chứa trong các bảng (table). Tuy nhiên bảng là một khái niệm mang tính hình tượng để chúng ta dễ suy nghĩ. Dữ liệu thật sự của mỗi bảng sẽ được chia nhỏ ra và chứa trong các trang (page) có kích thước bằng nhau.

Mỗi trang (page) được đặt cho 1 mã định danh gọi là buffertag. Buffertag này bao gồm 3 con số, mà thông qua nó sẽ định danh page này thuộc về table nào, và là page thứ mấy trong số các page đang chứa dữ liệu của table.

Dữ liệu trong page chính là dữ liệu thô sẽ được lưu trữ trong bộ nhớ tạm, cụ thể là buffer pool layer của buffer manager.

Quy trình hoạt động của Bộ quản lý vùng đệm



Khi 1 câu truy vấn SQL được gửi đến, các tiến trình sẽ xử lý logic của câu truy vấn và sẽ biết được dữ liệu cần thiết hiện đang được lưu ở trang nào. Hệ thống sẽ mang buffer_tag của trang tìm được đến kiểm tra với bộ quản lý đệm xem trang này đã có sẵn trong vùng đệm hay chưa.

Nếu trang chưa được lưu trong vùng đệm thì bộ quản lý sẽ yêu cầu truy xuất trực tiếp xuống đĩa cứng để lấy, sau đó lưu nó vào 1 trong các ô nhớ của vùng đệm. Vị trí của ô đó (buffer_id) sẽ được trả về cho tiến trình yêu cầu.

Ngược lại, nếu trang đang được lưu trong vùng đệm thì bộ quản lý sẽ trả về buffer_id của ô hiện đang chứa trang này.

Tiến trình sau khi nhận được buffer_id có thể thực hiện việc truy xuất trực tiếp dữ liệu mà nó cần.

Khi vùng đệm bị đầy

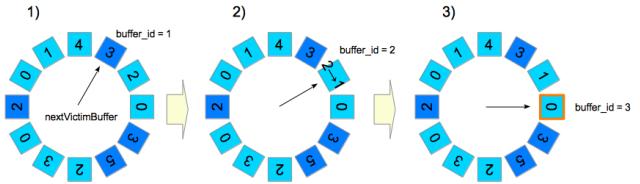
Buffer pool được lưu trữ trên RAM, 1 lúc nào đó nó cũng sẽ phải đầy. Do đó chúng ta cần 1 thuật toán để xác định nên bỏ đi trang nào để lấy chỗ cho trang mới.

Có nhiều thuật toán khác nhau để làm việc này. PostgreSQL từ bản 8.1 trở đi sử dụng thuật toán Clock Sweep thay cho LRU ở các phiên bản trước.

Ý tưởng của thuật toán Clock Sweep là: ưu tiên loại bỏ page nào ít được truy xuất nhất (usagecount), và tại thời gian xem xét, page đó đang không bị process nào truy xuất (unpinned_).

Thông tin về số lượng truy xuất, số lượng tiến trình đang truy xuất trang đó gọi là metadata của buffer slot đó, và

```
WHILE true
(1) Obtain the candidate buffer descriptor pointed by the nextVictimBuffer
(2) IF the candidate descriptor is unpinned THEN
     IF the candidate descriptor's usage_count == 0 THEN
        BREAK WHILE LOOP /* the corresponding slot of this descriptor is victim slot. */
     ELSE
        Decrease the candidate descriptor's usage_count by 1
     END IF
(4) Advance nextVictimBuffer to the next one
(5) RETURN buffer_id of the victim
```



được quản lý bởi Buffer descriptor.

Khi dữ liệu trong buffer pool bị thay đổi

Khi cơ sở dữ liệu nhận được 1 yêu cầu thực hiện thay đổi dữ liệu của bảng, những điều chỉnh sẽ được áp dụng trực tiếp lên trang tương ứng trong vùng đệm (chứ ko phải là phiên bản gốc ở đĩa cứng). Lúc này, trang tương ứng trong vùng đệm đã bị thay đổi sẽ được đánh dấu là bẩn (dirty).

Những thông tin mới này 1 lúc nào đó sẽ phải được lưu trữ cố định xuống đĩa cứng. Hành động này gọi là **Flush dirty page**.

Đảm nhận nhiệm vụ này sẽ là 2 tiến trình chạy nền là **checkpointer** và **background writer**.

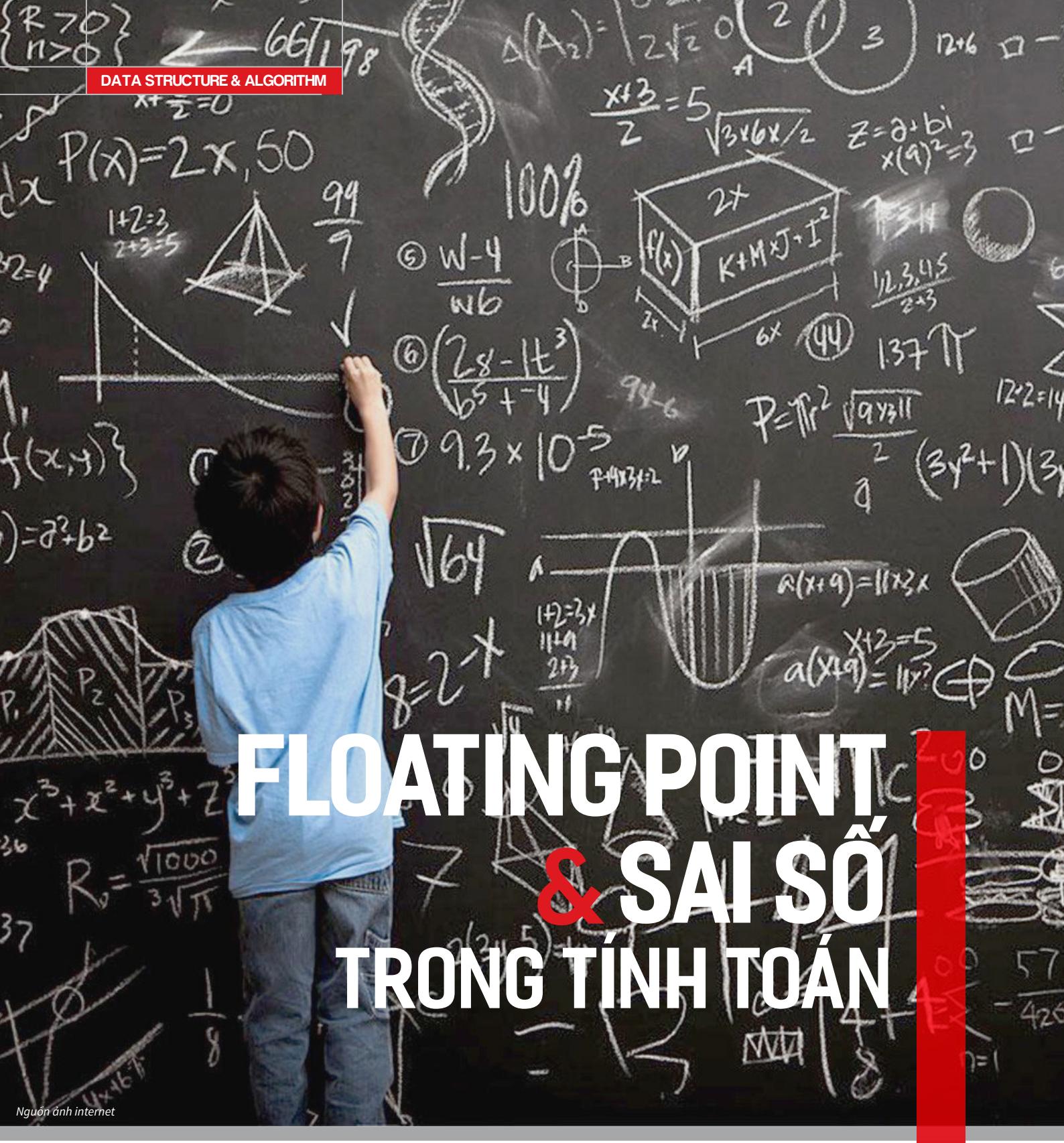
MỞ RỘNG

Như các bạn đã biết vùng đệm nằm trên RAM, dữ liệu của RAM sẽ mất hết nếu hệ thống xảy ra sự cố như mất điện. Vậy nếu trang trở nên bẩn (dirty), chưa kịp lưu cố định xuống đĩa cứng mà sự cố xảy ra thì đồng nghĩa là chúng ta sẽ mất những dữ liệu mới này. Điều này dẫn đến cần phải có 1 thành phần khác đảm nhiệm việc này, đó là WAL.

Mô tả chi tiết của WAL sẽ nằm ngoài phạm vi của bài viết này, nhưng nếu giải thích một cách ngắn gọn thì ý tưởng của WAL sẽ là: Nếu 1 hành động nào đó làm thay đổi trạng thái của dữ liệu, sự thay đổi đó sẽ được ghi nhận (log) lại trước cả khi buffered page được cập nhật. Như vậy, nếu sự cố xảy ra thì hệ thống sẽ xem xét lại các thông tin đã được ghi nhận lại (log) chưa được xử lý và do vậy sẽ đảm bảo dữ liệu được nguyên vẹn.



Đọc và thảo luận bài viết online:
<https://engineering.grokking.org/>



FLOATING POINT & SAI SỐ TRONG TÍNH TOÁN

Nguồn ảnh internet

Về tác giả:

PHÚ TRẦN - hiện đang công tác tại Axon Enterprise Inc. ở vị trí Research Engineer. Từng có 3 năm kinh nghiệm làm hệ thống nhúng tiêu tốn năng lượng thấp, trách nhiệm của mình hiện tại là làm sao chạy được các thuật toán AI trên hardware nhanh nhất và hiệu quả nhất. Sở thích cá nhân: chụp hình và đi du lịch để chụp hình!



Đọc và thảo luận bài viết online:
<https://engineering.grokking.org/>

TẠI SAO CHÚNG TA CẦN FLOATING POINT?

Với kiểu dữ liệu fixed point, dấu chấm thập phân (decimal point) là cố định:

$f < decimal, frac >$

trong đó: $decimal$ là số bit biểu thị phần nguyên, $frac$ là số bit biểu thị phần thập phân.

$$11011101_2 = 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = 27.625$$

Lấy ví dụ về một kiểu dữ liệu fixed point 8 bit, phần decimal là 5 bit và phần $frac$ là 3 bit. Ta có:

Kiểu dữ liệu **fixed point** như trên là rất đơn giản và phù hợp với những ứng dụng có yêu cầu độ chính xác tương đối. Nhưng đối với các ứng dụng yêu cầu độ chính xác rất cao hoặc rất thấp, nó sẽ gặp nhiều giới hạn và không sử dụng hiệu quả được các bit đang có.

Chính vì thế, chúng ta cần đến kiểu dữ liệu **floating point** - dấu chấm động - để biểu thị một số thực bằng mã nhị phân.

Tuy nhiên **floating point** cũng có hạn chế nhất định của nó, cái mà có thể gây ra lỗi không mong muốn trong quá trình coding. Trước khi bắt đầu, các bạn hãy thử xem đoạn code sau đây:

```
int main(int argc, char ** arg)
{
    float a = 1e10f;
    float b = 1e10f;
    float c = 0.1f;
    float result_1 = a - b + c;
    float result_2 = a + c - b;
    printf("a - b + c = %.4f\n", result_1);
    printf("a + c - b = %.4f\n", result_2);
    return 0;
}
```

Theo bạn, kết quả in ra sẽ là gì? Có thể nhiều bạn sẽ ngạc nhiên sau khi chạy thử chương trình:

```
a - b + c = 0.1000
a + c - b = 0.0000
```

Bài viết này sẽ làm rõ vấn đề tại sao kết quả lại là như trên.

ĐỊNH DẠNG CHẤM ĐỘNG THEO CHUẨN IEEE 754

Theo tiêu chuẩn IEEE 754 về số floating point, có tổng cộng 3 trường để biểu thị một số float, lần lượt là:

- Sign bit: bit dấu, luôn là 1 bit và là bit có trọng số cao nhất (most significant bit)
- Exponent bits: phần mũ, cơ số luôn là 2
- Fraction bits: phần thập phân, hay còn là mantissa

Sign	Exponent	Fraction
1 bit	m bits	n bits

Các thành phần của số floating point và số lượng bit

Tùy theo từng loại giá trị mà một số floating point muốn biểu thị, ta có mối liên hệ giữa 3 phần trên khác nhau:

- Normalized form
 - Sign: 0 hoặc 1
 - Exponent: bất kì số nào, ngoại trừ trường hợp $[11..1]_m$ hoặc $[00..0]_m$
 - Fraction: bất kì số nào
 - Số thực = $(-1)^{\text{sign}} \times 2^{\text{exponent}-\text{bias}} \times (0.F_1 F_2 F_3 ... F_n)$ trong đó bias = $2^{m-1} - 1$
- Denormalized form
 - Sign: 0 hoặc 1
 - Exponent: luôn luôn $[00..0]_m$
 - Fraction: bất kì số nào, ngoại trừ $[00..0]_n$
 - Số thực = $(-1)^{\text{sign}} \times 2^{1-\text{bias}} \times (0.F_1 F_2 F_3 ... F_n)$
- Zero
 - Sign: 0 hoặc 1, tương ứng 0^+ và 0^-
 - Exponent: luôn luôn $[00..0]_m$
 - Fraction: luôn luôn $[00..0]_n$
- Infinity
 - Sign: 0 hoặc 1, tương ứng $+\infty$ và $-\infty$
 - Exponent: luôn luôn $[11..1]_m$
 - Fraction: luôn luôn $[00..0]_n$
- NaN (not a number)
 - Sign: 0 hoặc 1
 - Exponent: luôn luôn $[11..1]_m$
 - Fraction: bất kì số nào, ngoại trừ $[00..0]_n$

Chúng ta đã thấy tiêu chuẩn IEEE 754 đã mã hóa một số thực bằng hệ nhị phân như thế nào. Dưới đây là một số kiểu dữ liệu floating point đã được định nghĩa trước và các thông số của chúng:

Kiểu dữ liệu	Tổng số lượng bit	Sign bit	Exponent bits	Fraction bits (Mantissa)	Bias	Số chữ số thập phân tối đa (*)
Half precision	16	1	5	10	15	3.31
Single precision	32	1	8	23	127	7.224
Double precision	64	1	11	52	1023	15.954

Một số kiểu floating point thông dụng được định nghĩa trong IEEE 754

(*): tối đa bao nhiêu số chữ số thập phân sau dấu phẩy có thể được biểu thị, hay nói cách khác là độ chính xác. Với n fraction bits, ta có được tối đa số chữ số thập phân được xác định bằng công thức: $\text{decimal digits} = \log_{10} 2^{n+1}$. Lấy ví dụ như chúng ta có số bit fraction $n = 1$, như vậy phần thập phân chúng ta có thể có tối đa là 1 chữ số. Nếu có 9 bit fraction, ta có độ chính xác đến 3 chữ số thập phân. Như vậy, càng nhiều

bit fraction, ta lại càng có độ chính xác cao hơn.

PHÉP LÀM TRÒN

Bởi vì kiểu dữ liệu floating point có hữu hạn số bit phần thập phân, nếu số thực có số chữ số thập phân lớn hơn số bit cho phép, những phần thập phân mà floating point không biểu thị được sẽ được bỏ đi, và số thực sẽ được làm tròn.

IEEE 754 quy định có 4 chế độ làm tròn khác nhau:

Round to nearest: chế độ mặc định, kết quả được làm tròn tới số floating point gần nhất. Nếu nằm giữa 2 số floating point thì sẽ chọn số có bit thấp nhất là 0.

Round toward 0: kết quả được làm tròn tới số floating point gần nhất và gần giá trị 0

Round toward +∞: hay còn gọi là ceiling, kết quả được làm tròn tới số floating point gần nhất và lớn hơn số thực

Round toward -∞: hay còn gọi là floor, kết quả được làm tròn tới số floating point gần nhất và nhỏ hơn số thực

Một số ví dụ cho thấy cách hoạt động của các chế độ làm tròn (giả sử ta làm tròn đến phần nguyên):

	+5.5	+6.25	-5.5	-6.25
round to nearest	+6	+6	-6	-6
round toward 0	+5	+6	-5	-6
round toward +∞	+6	+7	-5	-6
round toward -∞	+5	+6	-6	-7

Các kiểu làm tròn số float được quy định bởi IEEE 754

SAI SỐ DO PHÉP LÀM TRÒN

Như ta thấy, floating point không biểu thị chính xác một số thực, mà nó chỉ có thể xấp xỉ thông qua phép làm tròn. Tương tự như các kiểu dữ liệu khác, ta cũng có các phép toán cơ bản trên số float, ví dụ: cộng, trừ, nhân, chia, so sánh... và, không loại trừ, kết quả của các phép toán này cũng sẽ được làm tròn như vậy. Vì vậy, khi ta thay đổi thứ tự thực hiện các phép toán hoặc thay bằng phép toán tương đương nhau, kết quả mỗi trường hợp sẽ khác nhau.

Một ví dụ về việc thay đổi thứ tự thực hiện phép toán:

```
int main(int argc, char ** arg)
{
    float a = 0.2f;
    float b = 0.3f;
    float c = 0.4f;
    float sum_1 = (a + b) + c;
    float sum_2 = a + (b + c);
    printf("Sum_1 is %0.10f \\ Sum_2 is %0.10f\n", sum_1, sum_2);
    return 0;
}
```

Biên dịch đoạn code trên Linux bằng gcc:

```
gcc -o my_prog my_program.cpp -O0
```

Kết quả khi chạy chương trình:

Sum 1 is 0.8999999762 \ Sum 2 is 0.9000000358

Do các số float a, b, c đều là xấp xỉ của 0.2, 0.3, 0.4. Sự khác nhau giữa 2 kết quả đến từ sự khác nhau về thứ tự làm tròn:

$$\begin{aligned}a &= 2^{-3} \times 1.100110011001100110011001101_2 \\b &= 2^{-2} \times 1.00110011001100110011001101_2 \\c &= 2^{-2} \times 1.100110011001100110011001101_2\end{aligned}$$

Trường hợp $(a + b) + c$:

```

a + b = 2-1 × 1.000000000000000000000000000010002

round(a + b)

= 2-1 × 1.00000000000000000000000000002

round(round(a + b) + c)

= 2-1 × 1.110011001100110011001102

```

Trường hợp a + (b + c):

$$\begin{aligned}
 \mathbf{b} + \mathbf{c} &= 2^{-1} \times 1.011001100110011001100110000_2 \\
 \text{round}(\mathbf{b} + \mathbf{c}) &= 2^{-1} \times 1.01100110011001100110100_2 \\
 \text{round}(\mathbf{a} + \text{round}(\mathbf{b} + \mathbf{c})) &= 2^{-1} \times 1.11001100110011001110000_2
 \end{aligned}$$

Một ví dụ khác về việc thay đổi phép toán tương đương cũng có thể làm khác kết quả:

```
int main(int argc, char ** arg)
{
    float a = 0.1f;
    float sum = 0;
    for(int i = 0; i < 10; i++)
    {
        sum += a;
    }
    float product = a * 10;
    printf("Sum is %0.10f \\ Product is %0.10f
\n", sum,
           product);
    return      0;
}
```

Kết quả sau khi biên dịch:

Sum is 1.0000001192 \ Product is 1.0000000000

Ở đây, sum được tính bằng cách cộng với a 10 lần, tức là trong đó có 10 lần làm tròn số. Còn đối với product, kết quả chỉ được làm tròn 1 lần sau phép nhân 10, dẫn đến ít sai số hơn so với sum.

PHÉP TÍNH FUSED MULTIPLY-ADD VÀ ỨNG DỤNG

Để làm giảm sai số do phép làm tròn, phép toán fused multiply-add (FMA) được thêm vào IEEE754 vào năm 2008. Bản chất của FMA là chỉ làm tròn 1 lần sau kết quả nhân-

công:

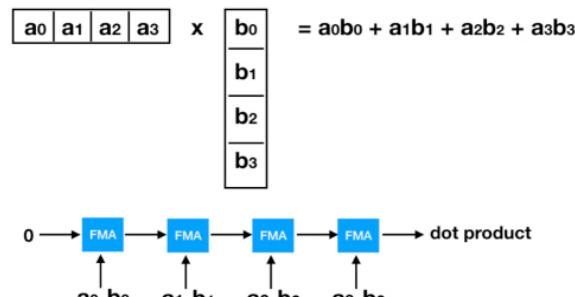
$$FMA(X, Y, Z) = \text{round}(XY + Z)$$

Một điểm cộng với phép FMA là do chỉ làm tròn 1 lần ở kết quả cuối nên nó sẽ nhanh hơn nếu ta dùng phép nhân và cộng bình thường.

Trên thị trường hiện nay đã có các dòng CPU và GPU hỗ trợ phép FMA trực tiếp, đi kèm là các tập lệnh FMA cho một phép tính hoặc cho lệnh SIMD (single instruction, multiple data):

- AMD: từ dòng Bulldozer (2011), Piledriver(2012),...
 - Intel: từ dòng Haswell (2013), Broadwell,...
 - ARM: ARM Cortex M4F, ARM Cortex A5, ARM Cortex A7,...
 - NVIDIA: các GPUs có kiến trúc từ Fermi (2010) trở về sau

Một ứng dụng quan trọng của FMA, đó là tăng tốc phép tính nhân ma trận:



Phép nhận ma trận dùng FMA

TRẢ LỜI CÂU HỎI MỞ ĐẦU

Đến đây, chắc các bạn cũng đã nắm rõ khái quát floating point là gì, cách nó xấp xỉ một số thực như sao, cũng như sai số tiềm ẩn trong floating point mà chúng ta cần lưu ý khi code. Từ đó chúng ta có thể giải thích câu hỏi được đưa ra đầu bài như sau:

$$\begin{aligned}a &= 2^{33} \times 1.00101010000001011111001_2 \\b &= 2^{33} \times 1.00101010000001011111001_2 \\c &= 2^{-4} \times 1.10011001100110011001101_2\end{aligned}$$

Trường hợp a - b + c:

Trường hợp a + c - b:

THIẾT KẾ TÍNH NĂNG & FEATURE TOGGLE FEATURE ROLLOUT



Bạn sẽ làm gì khi muốn triển khai một tính năng đến một nhóm nhỏ người dùng thay vì toàn bộ người dùng trong hệ thống? | **HUY NGUYỄN**

- ▶ **Feature Toggle** (hay *Feature Flags*) và **Feature Rollout** là một kĩ thuật phổ biến giúp bạn có thể quản lý được hành vi của phần mềm của mình mà không cần phải đổi code & deploy lại. Trong bài viết này mình sẽ đi qua một vài tính năng từ cơ bản đến nâng cao, đồng thời chia sẻ cách tiếp cận để design và implement một hệ thống Feature Toggle/Feature Rollout nhỏ.

Một vài ứng dụng trong thực tế của kỹ thuật này:

Feature Control: tắt/bật tính năng cho toàn bộ hệ thống, hoặc một vài khách hàng cụ thể

Feature Rollout: bật một tính năng từ từ cho tập người dùng của mình, có thể trước mắt cho 1 vài người dùng cụ thể, rồi sau đó bật tiếp cho những người dùng trong một nước/khu vực nào đó.

BẬT/TẮT TOÀN CỤC

Trường hợp cơ bản nhất của Feature Toggle là giúp bạn bật/tắt 1 tính năng mà không cần phải deploy lại code. Ví dụ, chúng ta có tính năng XYZ như sau:

```
if FeatureToggle.check('xyz')
    # feature xyz is turned on, do what you need here
end
```

```
# code minh họa bằng Ruby
class FeatureToggle
    def self.check (feature_key)
        feature = FeatureToggle.get(feature_key)
        return feature&.toggled == true
    end
end
```

Lúc này, bạn có thể code & deploy tính năng xyz đó trước, sau đó chọn thời điểm thích hợp để bật nó lên cho toàn bộ hệ thống. Hoặc ngược lại, khi bạn phát hiện ra lỗi, hoặc hệ thống quá tải, bạn có thể tắt bớt 1 vài tính năng

phụ chỉ với 1 vài cú click chuột.

Thiết kế đãng sau đơn giản là một bảng trong cơ sở dữ liệu chứa các tính năng của bạn, cùng với 1 biến cờ quyết định trạng thái bật/tắt của tính năng.

```
CREATE TABLE feature_toggles (
    feature_key VARCHAR(255) PRIMARY KEY,
    toggled BOOLEAN DEFAULT FALSE
);
```

BẬT/TẮT CHO KHÁCH HÀNG CỤ THỂ

Bây giờ mình thảo luận đến một vài trường hợp phức tạp hơn khi mà bạn:

1. Chỉ muốn bật tính năng cho 1 vài khách hàng thử nghiệm (hoặc là khách hàng cao cấp)
2. Muốn bật cho toàn bộ hệ thống, nhưng tắt cho 1 vài khách hàng đặc biệt (khách hàng chưa đăng ký sử dụng tính năng này)

```
if FeatureToggle.check('xyz', current_customer_id)
  # feature xyz is turned on for this customer
end
```

Có thể nhận ra rằng 2 trường hợp trên giống như 2 mệnh đề đảo ngược nhau, lúc đó, bạn có thể cải thiện hệ thống bằng cách thêm vào 1 cột exception_list để chứa tập khách hàng đặc biệt này:

```
CREATE TABLE feature_toggles (
    feature_key VARCHAR(255) PRIMARY KEY,
    toggled BOOLEAN DEFAULT FALSE,
    exception_list JSONB DEFAULT '[]'
);
```

Lúc này thì:

`toggled = false, exception_list = [1, 2]`: tắt toàn bộ, vào bật cho khách hàng ID 1 và 2.

`toggled = true, exception_list = [1, 2]`: bật cho toàn bộ khách hàng, chỉ tắt cho khách hàng ID 1 và 2.

Batch Actions						
#	Key	Notes	Toggle Mode	Except Tenant Ids	Updated At	
19	special_eventlet_it_snow	Many Christmas everyone!	enabled		December 20, 2017 04:25	View Edit Delete
18	email_schedule_fresh_report_data	Allow email schedules to bypass cache	disabled	+ 41	December 20, 2017 04:25	View Edit Delete
17	report_direct_export	Direct Export feature inside report. [5, 7, 52, 26]	enabled		December 20, 2017 04:26	View Edit Delete
16	schema_synchronization		disabled		December 08, 2017 20:00	View Edit Delete
14	billing_x2		enabled	+ 175 + 414 + 415 + 416 + 417 + 418 + 419 + 420 + 421 + 422 + 423 + 424 + 425 + 426 + 427 + 428 + 429 + 430 + 431 + 432 + 433 + 434 + 435 + 436 + 437 + 438 + 439 + 440 + 441 + 442 + 443 + 444 + 445 + 446 + 447 + 448 + 449 + 450 + 451 + 452 + 453 + 454 + 455 + 456 + 457 + 458 + 459 + 460 + 461 + 462 + 463 + 464 + 465 + 466 + 467 + 468 + 469 + 470 + 471 + 472 + 473 + 474 + 475 + 476 + 477 + 478 + 479 + 480 + 481 + 482 + 483 + 484 + 485 + 486 + 487 + 488 + 489 + 490 + 491 + 492 + 493 + 494 + 495 + 496 + 497 + 498 + 499 + 500 + 501 + 502 + 503 + 504 + 505 + 506 + 507 + 508 + 509 + 510 + 511 + 512 + 513 + 514 + 515 + 516 + 517 + 518 + 519 + 520 + 521 + 522 + 523 + 524 + 525 + 526 + 527 + 528 + 529 + 530 + 531 + 532 + 533 + 534 + 535 + 536 + 537 + 538 + 539 + 540 + 541 + 542 + 543 + 544 + 545 + 546 + 547 + 548 + 549 + 550 + 551 + 552 + 553 + 554 + 555 + 556 + 557 + 558 + 559 + 560 + 561 + 562 + 563 + 564 + 565 + 566 + 567 + 568 + 569 + 570 + 571 + 572 + 573 + 574 + 575 + 576 + 577 + 578 + 579 + 580 + 581 + 582 + 583 + 584 + 585 + 586 + 587 + 588 + 589 + 590 + 591 + 592 + 593 + 594 + 595 + 596 + 597 + 598 + 599 + 600 + 601 + 602 + 603 + 604 + 605 + 606 + 607 + 608 + 609 + 610 + 611 + 612 + 613 + 614 + 615 + 616 + 617 + 618 + 619 + 620 + 621 + 622 + 623 + 624 + 625 + 626 + 627 + 628 + 629 + 630 + 631 + 632 + 633 + 634 + 635 + 636 + 637 + 638 + 639 + 640 + 641 + 642 + 643 + 644 + 645 + 646 + 647 + 648 + 649 + 650 + 651 + 652 + 653 + 654 + 655 + 656 + 657 + 658 + 659 + 660 + 661 + 662 + 663 + 664 + 665 + 666 + 667 + 668 + 669 + 670 + 671 + 672 + 673 + 674 + 675 + 676 + 677 + 678 + 679 + 680 + 681 + 682 + 683 + 684 + 685 + 686 + 687 + 688 + 689 + 690 + 691 + 692 + 693 + 694 + 695 + 696 + 697 + 698 + 699 + 700 + 701 + 702 + 703 + 704 + 705 + 706 + 707 + 708 + 709 + 710 + 711 + 712 + 713 + 714 + 715 + 716 + 717 + 718 + 719 + 720 + 721 + 722 + 723 + 724 + 725 + 726 + 727 + 728 + 729 + 730 + 731 + 732 + 733 + 734 + 735 + 736 + 737 + 738 + 739 + 740 + 741 + 742 + 743 + 744 + 745 + 746 + 747 + 748 + 749 + 750 + 751 + 752 + 753 + 754 + 755 + 756 + 757 + 758 + 759 + 760 + 761 + 762 + 763 + 764 + 765 + 766 + 767 + 768 + 769 + 770 + 771 + 772 + 773 + 774 + 775 + 776 + 777 + 778 + 779 + 780 + 781 + 782 + 783 + 784 + 785 + 786 + 787 + 788 + 789 + 790 + 791 + 792 + 793 + 794 + 795 + 796 + 797 + 798 + 799 + 800 + 801 + 802 + 803 + 804 + 805 + 806 + 807 + 808 + 809 + 810 + 811 + 812 + 813 + 814 + 815 + 816 + 817 + 818 + 819 + 820 + 821 + 822 + 823 + 824 + 825 + 826 + 827 + 828 + 829 + 830 + 831 + 832 + 833 + 834 + 835 + 836 + 837 + 838 + 839 + 840 + 841 + 842 + 843 + 844 + 845 + 846 + 847 + 848 + 849 + 850 + 851 + 852 + 853 + 854 + 855 + 856 + 857 + 858 + 859 + 860 + 861 + 862 + 863 + 864 + 865 + 866 + 867 + 868 + 869 + 870 + 871 + 872 + 873 + 874 + 875 + 876 + 877 + 878 + 879 + 880 + 881 + 882 + 883 + 884 + 885 + 886 + 887 + 888 + 889 + 890 + 891 + 892 + 893 + 894 + 895 + 896 + 897 + 898 + 899 + 900 + 901 + 902 + 903 + 904 + 905 + 906 + 907 + 908 + 909 + 910 + 911 + 912 + 913 + 914 + 915 + 916 + 917 + 918 + 919 + 920 + 921 + 922 + 923 + 924 + 925 + 926 + 927 + 928 + 929 + 930 + 931 + 932 + 933 + 934 + 935 + 936 + 937 + 938 + 939 + 940 + 941 + 942 + 943 + 944 + 945 + 946 + 947 + 948 + 949 + 950 + 951 + 952 + 953 + 954 + 955 + 956 + 957 + 958 + 959 + 960 + 961 + 962 + 963 + 964 + 965 + 966 + 967 + 968 + 969 + 970 + 971 + 972 + 973 + 974 + 975 + 976 + 977 + 978 + 979 + 980 + 981 + 982 + 983 + 984 + 985 + 986 + 987 + 988 + 989 + 990 + 991 + 992 + 993 + 994 + 995 + 996 + 997 + 998 + 999 + 1000 + 1001 + 1002 + 1003 + 1004 + 1005 + 1006 + 1007 + 1008 + 1009 + 1010 + 1011 + 1012 + 1013 + 1014 + 1015 + 1016 + 1017 + 1018 + 1019 + 1020 + 1021 + 1022 + 1023 + 1024 + 1025 + 1026 + 1027 + 1028 + 1029 + 1030 + 1031 + 1032 + 1033 + 1034 + 1035 + 1036 + 1037 + 1038 + 1039 + 1040 + 1041 + 1042 + 1043 + 1044 + 1045 + 1046 + 1047 + 1048 + 1049 + 1050 + 1051 + 1052 + 1053 + 1054 + 1055 + 1056 + 1057 + 1058 + 1059 + 1060 + 1061 + 1062 + 1063 + 1064 + 1065 + 1066 + 1067 + 1068 + 1069 + 1070 + 1071 + 1072 + 1073 + 1074 + 1075 + 1076 + 1077 + 1078 + 1079 + 1080 + 1081 + 1082 + 1083 + 1084 + 1085 + 1086 + 1087 + 1088 + 1089 + 1090 + 1091 + 1092 + 1093 + 1094 + 1095 + 1096 + 1097 + 1098 + 1099 + 1100 + 1101 + 1102 + 1103 + 1104 + 1105 + 1106 + 1107 + 1108 + 1109 + 1110 + 1111 + 1112 + 1113 + 1114 + 1115 + 1116 + 1117 + 1118 + 1119 + 1120 + 1121 + 1122 + 1123 + 1124 + 1125 + 1126 + 1127 + 1128 + 1129 + 1130 + 1131 + 1132 + 1133 + 1134 + 1135 + 1136 + 1137 + 1138 + 1139 + 1140 + 1141 + 1142 + 1143 + 1144 + 1145 + 1146 + 1147 + 1148 + 1149 + 1150 + 1151 + 1152 + 1153 + 1154 + 1155 + 1156 + 1157 + 1158 + 1159 + 1160 + 1161 + 1162 + 1163 + 1164 + 1165 + 1166 + 1167 + 1168 + 1169 + 1170 + 1171 + 1172 + 1173 + 1174 + 1175 + 1176 + 1177 + 1178 + 1179 + 1180 + 1181 + 1182 + 1183 + 1184 + 1185 + 1186 + 1187 + 1188 + 1189 + 1190 + 1191 + 1192 + 1193 + 1194 + 1195 + 1196 + 1197 + 1198 + 1199 + 1200 + 1201 + 1202 + 1203 + 1204 + 1205 + 1206 + 1207 + 1208 + 1209 + 1210 + 1211 + 1212 + 1213 + 1214 + 1215 + 1216 + 1217 + 1218 + 1219 + 1220 + 1221 + 1222 + 1223 + 1224 + 1225 + 1226 + 1227 + 1228 + 1229 + 1230 + 1231 + 1232 + 1233 + 1234 + 1235 + 1236 + 1237 + 1238 + 1239 + 1240 + 1241 + 1242 + 1243 + 1244 + 1245 + 1246 + 1247 + 1248 + 1249 + 1250 + 1251 + 1252 + 1253 + 1254 + 1255 + 1256 + 1257 + 1258 + 1259 + 1260 + 1261 + 1262 + 1263 + 1264 + 1265 + 1266 + 1267 + 1268 + 1269 + 1270 + 1271 + 1272 + 1273 + 1274 + 1275 + 1276 + 1277 + 1278 + 1279 + 1280 + 1281 + 1282 + 1283 + 1284 + 1285 + 1286 + 1287 + 1288 + 1289 + 1290 + 1291 + 1292 + 1293 + 1294 + 1295 + 1296 + 1297 + 1298 + 1299 + 1300 + 1301 + 1302 + 1303 + 1304 + 1305 + 1306 + 1307 + 1308 + 1309 + 1310 + 1311 + 1312 + 1313 + 1314 + 1315 + 1316 + 1317 + 1318 + 1319 + 1320 + 1321 + 1322 + 1323 + 1324 + 1325 + 1326 + 1327 + 1328 + 1329 + 1330 + 1331 + 1332 + 1333 + 1334 + 1335 + 1336 + 1337 + 1338 + 1339 + 1340 + 1341 + 1342 + 1343 + 1344 + 1345 + 1346 + 1347 + 1348 + 1349 + 1350 + 1351 + 1352 + 1353 + 1354 + 1355 + 1356 + 1357 + 1358 + 1359 + 1360 + 1361 + 1362 + 1363 + 1364 + 1365 + 1366 + 1367 + 1368 + 1369 + 1370 + 1371 + 1372 + 1373 + 1374 + 1375 + 1376 + 1377 + 1378 + 1379 + 1380 + 1381 + 1382 + 1383 + 1384 + 1385 + 1386 + 1387 + 1388 + 1389 + 1390 + 1391 + 1392 + 1393 + 1394 + 1395 + 1396 + 1397 + 1398 + 1399 + 1400 + 1401 + 1402 + 1403 + 1404 + 1405 + 1406 + 1407 + 1408 + 1409 + 1410 + 1411 + 1412 + 1413 + 1414 + 1415 + 1416 + 1417 + 1418 + 1419 + 1420 + 1421 + 1422 + 1423 + 1424 + 1425 + 1426 + 1427 + 1428 + 1429 + 1430 + 1431 + 1432 + 1433 + 1434 + 1435 + 1436 + 1437 + 1438 + 1439 + 1440 + 1441 + 1442 + 1443 + 1444 + 1445 + 1446 + 1447 + 1448 + 1449 + 1450 + 1451 + 1452 + 1453 + 1454 + 1455 + 1456 + 1457 + 1458 + 1459 + 1460 + 1461 + 1462 + 1463 + 1464 + 1465 + 1466 + 1467 + 1468 + 1469 + 1470 + 1471 + 1472 + 1473 + 1474 + 1475 + 1476 + 1477 + 1478 + 1479 + 1480 + 1481 + 1482 + 1483 + 1484 + 1485 + 1486 + 1487 + 1488 + 1489 + 1490 + 1491 + 1492 + 1493 + 1494 + 1495 + 1496 + 1497 + 1498 + 1499 + 1500 + 1501 + 1502 + 1503 + 1504 + 1505 + 1506 + 1507 + 1508 + 1509 + 1510 + 1511 + 1512 + 1513 + 1514 + 1515 + 1516 + 1517 + 1518 + 1519 + 1520 + 1521 + 1522 + 1523 + 1524 + 1525 + 1526 + 1527 + 1528 + 1529 + 1530 + 1531 + 1532 + 1533 + 1534 + 1535 + 1536 + 1537 + 1538 + 1539 + 1540 + 1541 + 1542 + 1543 + 1544 + 1545 + 1546 + 1547 + 1548 + 1549 + 1550 + 1551 + 1552 + 1553 + 1554 + 1555 + 1556 + 1557 + 1558 + 1559 + 1560 + 1561 + 1562 + 1563 + 1564 + 1565 + 1566 + 1567 + 1568 + 1569 + 1570 + 1571 + 1572 + 1573 + 1574 + 1575 + 1576 + 1577 + 1578 + 1579 + 1580 + 1581 + 1582 + 1583 + 1584 + 1585 + 1586 + 1587 + 1588 + 1589 + 1590 + 1591 + 1592 + 1593 + 1594 + 1595 + 1596 + 1597 + 1598 + 1599 + 1600 + 1601 + 1602 + 1603 + 1604 + 1605 + 1606 + 1607 + 1608 + 1609 + 1610 + 1611 + 1612 + 1613 + 1614 + 1615 + 1616 + 1617 + 1618 + 1619 + 1620 + 1621 + 1622 + 1623 + 1624 + 1625 + 1626 + 1627 + 1628 + 1629 + 1630 + 1631 + 1632 + 1633 + 1634 + 1635 + 1636 + 1637 + 1638 + 1639 + 1640 + 1641 + 1642 + 1643 + 1644 + 1645 + 1646 + 1647 + 1648 + 1649 + 1650 + 1651 + 1652 + 1653 + 1654 + 1655 + 1656 + 1657 + 1658 + 1659 + 1660 + 1661 + 1662 + 1663 + 1664 + 1665 + 1666 + 1667 + 1668 + 1669 + 1670 + 1671 + 1672 + 1673 + 1674 + 1675 + 1676 + 1677 + 1678 + 1679 + 1680 + 1681 + 1682 + 1683 + 1684 + 1685 + 1686 + 1687 + 1688 + 1689 + 1690 + 1691 		

GROKKING'S CORNER

KĨ SƯ PHẦN MỀM GIỎI THÌ PHẢI NHƯ THẾ NÀO?

■ NGUYỄN VĂN QUANG HUY

Sau một thời gian làm kĩ sư phần mềm ở nhiều môi trường, cũng như qua trải nghiệm phỏng vấn & tiếp xúc với rất nhiều bạn lập trình viên khác, có một nhận định chủ quan mà mình muốn đưa ra như sau:

Người làm phần mềm có 3 loại:

1- Người làm nghiên cứu (Computer Scientist) - thường làm việc ở trường ĐH và các viện nghiên cứu, thường có Master hoặc PhD. Kiến thức chuyên sâu của họ về khoa học máy tính rất tốt, tư duy giải quyết vấn đề cao. Họ thường không tham gia lập trình các dự án phần mềm lớn nên code thường không đẹp và sẽ gây nhiều khó khăn cho dự án, nhưng nếu trọng tâm của họ là nghiên cứu chuyên sâu thì không vấn đề gì cả.

2- Lập trình viên ứng dụng (Application Developer) - thường chuyên viết ứng dụng web, mobile, tuy nhiên họ thường thiếu các kiến thức về khoa học máy tính, thuật toán, cấu trúc dữ liệu. Một phần có thể do các bạn không có điều kiện được tiếp xúc vì ít người dạy, một phần thì do hầu hết tính chất công việc đang làm ở Việt Nam không cho các bạn cảm thấy những cái này quan trọng.

3- Kĩ sư phần mềm (Software Engineer) - là người có cái nhìn tổng quát tốt về mọi thứ, vừa có kiến thức vững về khoa học máy tính như người làm nghiên cứu, vừa có khả năng lập trình cực tốt. Đây là nhóm mà các công ty phần



mềm như Google, Facebook, Twitter luôn muốn tuyển vào.

Người làm nghiên cứu thì tiếp tục nghiên cứu vì đó là đam mê của họ. Còn người ở nhóm Application Developer nên/cần phải phát triển lên được trình độ của Software Engineer, vì đó đơn giản là con đường phát triển mà các bạn phải lên tới được.

Theo cảm nhận chủ quan mình thấy thì Việt Nam mình có nhiều người trong nhóm lập trình viên ứng dụng, nhưng rất ít bạn ở mức độ kĩ sư phần mềm. Do vậy, nhìn ngoài vào thì ai cũng nói ở Việt Nam có nhiều lập trình viên, nhưng đến khi vào phỏng vấn thì rất khó kiểm được bạn vừa ý.

(Lưu ý: Tên đặt ra có tính chất khái quát hoá, ko hàm chỉ chức danh của các bạn trong công việc hiện tại)

COMPUTER SCIENTIST

- Strong computer science knowledge
- Lack software development skills
- Work for research institutes

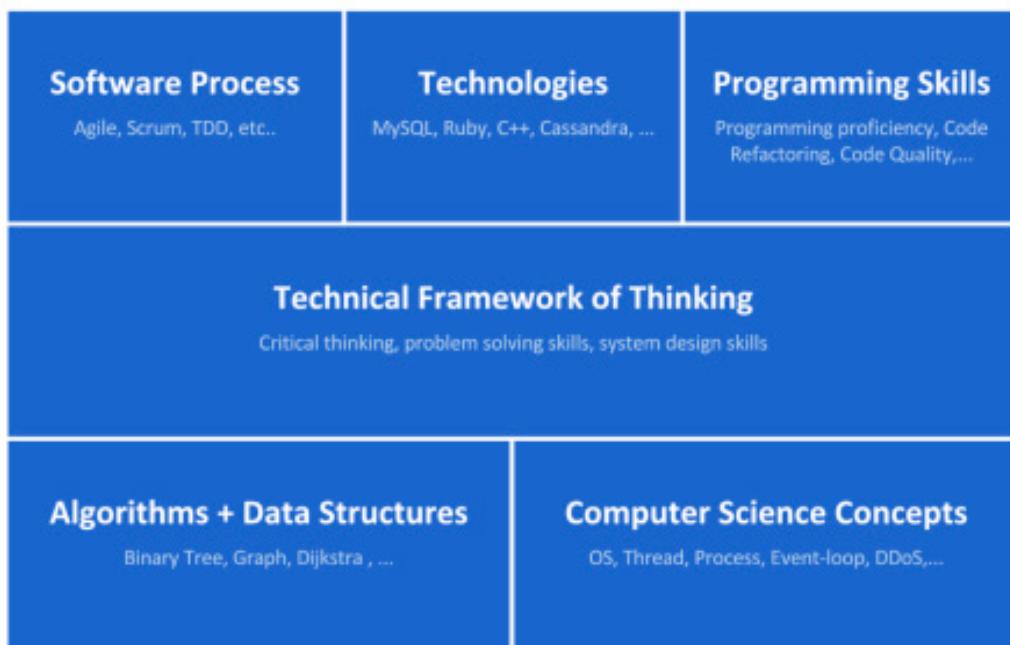
SOFTWARE ENGINEER

- Strong computer science foundations
- Strong software development skills
- Usually work for internet/ product companies

APPLICATION DEVELOPER

- Lack computer science knowledge
- Good software development skills
- Usually work for software development firms





VÂY TRỞ THÀNH KĨ SƯ PHẦN MỀM GIỎI (GOOGLE/FACEBOOK ENGINEER LEVEL) THÌ CẦN NHỮNG GÌ?

Đây là 1 biểu đồ các kiến thức nền tảng (fundamentals) mà mình tổng quát ra được, nhằm giúp các bạn định vị tốt hơn những điều các bạn đang có/cần bổ sung.

Nửa trên là những kiến thức thực tế cần thiết khi đi làm, bao gồm hiểu công nghệ, hiểu quy trình phát triển phần mềm và kĩ thuật lập trình (clean coding, code refactoring, v.v)

Nửa dưới là những kiến thức gốc về khoa học máy tính mà các bạn cần nắm. Cần lưu ý là mình muốn nhấn mạnh tầm quan trọng của tư duy kĩ thuật có hệ thống (technical framework of thinking), hay một cách khác là tư duy giải quyết vấn đề (problem solving), tư duy thiết kế hệ thống (system design). Tư duy này là khả năng kết nối những kiến thức lẻ độc lập (lego blocks) mà các bạn học được bên dưới với nhau để đưa ra hướng giải quyết vấn đề.

Để ý thấy là thường các bạn lập trình viên ứng dụng sẽ tập trung vào nửa trên, và các bạn nghiên cứu sẽ tập trung vào nửa dưới của biểu đồ trên.

Thêm nữa, ở Việt Nam nhiều bạn nghĩ tư duy hệ thống (system design/system architecture) là 1 vị trí độc lập, nâng cao kinh nghiệm một hối mới cần tới. Quan điểm của mình thì đây là tư duy mà một kĩ sư phần mềm nào đều cần phải có, nếu chưa có thì bạn chỉ ở mức thợ thô sơ chứ chưa phải là kĩ sư. Nếu bạn phỏng vấn vào các vị trí lập trình viên của Facebook/Google, bạn sẽ thấy những câu hỏi phỏng vấn đều sẽ kiểm tra tổng quát hết những kiến thức nêu trên (trừ khi bạn là sinh viên mới ra trường, lúc đó câu hỏi phỏng vấn sẽ xoay quanh lập trình thuật toán & cấu trúc dữ liệu).

Bạn có và chưa có gì trong những phần nhỏ của biểu đồ trên?



Sự hình thành và phát triển của các cộng đồng công nghệ tại Việt Nam

GROKKING TEAM



Dứng trước giai đoạn được xem là bước khởi đầu của cuộc cách mạng công nghiệp 4.0, chúng ta có thể dễ dàng nhận thấy Việt Nam đang tiếp cận những cơ hội lớn, sẵn sàng cho sự chuyển mình đầy tích cực và nhanh chóng trên toàn diện, không chỉ kinh tế, mà cả văn hóa, xã hội, giáo dục và công nghệ. Đáng kể và dễ dàng nhận thấy nhất trong lĩnh vực công nghệ thông tin chính là việc hình thành và phát triển các hội, nhóm, tổ chức mang tính cộng đồng và có sự kết nối cao những năm gần đây. Sự hình thành này bắt đầu vào năm 2012, vừa đem tới những lợi ích, hy vọng cho cộng đồng công nghệ, vừa bộc lộ nhiều điểm yếu, khó khăn. Nhưng cho tới bây giờ, người viết cho rằng đây mới chỉ là khởi đầu cho một chuyển động mạnh mẽ và tiềm năng hơn nữa, hứa hẹn là một động lực giúp thúc đẩy sự năng động và sáng tạo trong lĩnh vực công nghệ tại Việt Nam.

Bài viết này được dựa trên quan sát cá nhân của người viết, vốn đã từng tổ chức hoặc tham dự vào các cộng đồng công nghệ từ năm 2012 tới thời điểm hiện tại (năm 2018).



meetup.com chú trọng vào các hoạt động offline, yêu cầu các thành viên tham gia phải gặp mặt trực tiếp và giao lưu tương tác nhóm với nhau tại một địa điểm cụ thể. Việc meet-up xuất hiện tại Việt Nam gắn khá chặt với việc một số lập

trình viên người nước ngoài khởi xướng các hoạt động cộng đồng vào năm 2012. Một ví dụ điển hình là JS HCM meetup, sáng lập bởi Nicolas Embleton, một người nước ngoài đang ở VN. Sau đó, lần lượt iOS, Ruby meetup cũng ra đời trong giai đoạn này, với sự tham dự của nhiều lập trình viên ngoại quốc, hay là người Việt đang công tác tại nước ngoài (**Ruby Vietnam**).

Lịch sử hình thành và phát triển

2012-2013, những meetups đầu tiên xuất hiện, tạo tiền đề cho các hoạt động giao lưu trực tiếp (offline)

Dựa vào một vài thông tin về các hoạt động của những meet-up mà bản thân người viết đã từng tham gia trong thời gian này, 2012-2013 được

Có thể nhận thấy một vài đặc điểm về đặc thù của giai đoạn này:



Nội dung đơn giản và chưa có chiều sâu. Các hoạt động thường thấy của các meetup thời điểm này đều chủ yếu xoay quanh người chia sẻ (speakers). Các bài nói trong giai đoạn này hầu hết đều sơ sài và mang tính phi hình thức. Hầu hết diễn giả là những người đến từ mạng lưới quan hệ cá nhân của các thành viên ban tổ chức.Thêm vào đó, chiếm một phần không nhỏ các thành viên đến tham gia meetup với mục đích chính là để trau dồi ngoại ngữ, thay vì chuyên môn.

Quy mô nhỏ. Hầu hết quy mô của các meetup chỉ dao động ở quy mô từ 1-30 người. Địa điểm tổ chức thường là ở các quán cà phê nhỏ.

Mang tính tự phát và không bền vững. Hầu hết các meetup giai đoạn này đều được thành lập dựa trên yếu tố quen thân của một nhóm người làm nền tảng. Phần lớn các thành viên sẽ biết nhau từ trước, hoặc sẽ dần quen thuộc với nhau thông qua các hoạt động.

Chưa có nhân lực tổ chức bài bản. Các hoạt động giai đoạn này đa số đều là do chính bản thân các bạn lập trình viên đứng ra tự tổ chức nên hầu như đều không có yếu tố bài bản như các công ty sự kiện chuyên nghiệp cũng như không có mô hình truyền thông hiệu quả đến cho cộng đồng. Đó là lý do các meetup đa phần đều hoạt động cầm chừng với quy mô nhỏ lẻ và mang tính rời rạc, thiếu sự gắn kết.

Ngoài ra, nhắc đến sự phát triển của meetup thì không thể không kể đến vai trò của các không gian

làm việc chung (co-working space). Một trong những co-working space đầu tiên ở Sài Gòn được biết đến chính là SaigonHub. Việc hình thành các co-working space có thể nói là tiền đề cho sự phát triển mạnh của các meetup ngay sau đó. Vốn dĩ, bản thân co-working space là một mô hình mới, nơi cung cấp không gian để làm việc cho cá nhân hoặc các văn phòng của công ty startup vừa và nhỏ. Để co-working space tồn tại, người sáng lập phải phối hợp tổ chức thêm nhiều hoạt động/ sự kiện, hoặc cho thuê không gian sự kiện với giá rẻ hoặc thậm chí miễn phí. Chính điều này đã vô tình khiến cho các hoạt động của các meetup - vốn do những người tổ chức không có chuyên môn tổ chức sự kiện - diễn ra dễ dàng và thường xuyên hơn.

Cũng giống như các mô hình tiên phong khác, SaigonHub sau khi đổi mới với nhiều khó khăn như chi phí cao, thị trường vẫn chưa kịp đón nhận, chưa có nhiều sự kiện được tận dụng và tổ chức, mô hình còn mớiCuối cùng, nhóm sáng lập của SaigonHub đã quyết định đóng cửa co-working space này vào tháng 4/2014, một điều đáng tiếc. Tuy không thể tiếp tục hoạt động nhưng với sự hình thành của mình, SaigonHub đã tạo nên một nền tảng không nhỏ cho các co-working space ngày càng quy mô và chất lượng hơn hoạt động sau này.

2014 - đầu 2016: phát triển

Sang năm 2014, những mô hình meetup đầu tiên vẫn tiếp tục được duy trì. Tuy nhiên, một số khác phải dừng lại do thiếu cơ chế hoạt động bài bản và có tính tổ chức. Đầu vậy, 2014-2015 có thể được xem là hai năm chứng kiến sự chuyển mình tích cực dần của cộng đồng.

Có hai nguyên khách quan dẫn đến sự phát triển này. Đầu tiên là về yếu tố con người, những cá nhân tham gia tổ chức các meetup giai đoạn đầu dần trở nên trưởng thành hơn, trong đó có một vài cá nhân xuất sắc. Kế đến, những công

ty nhận thấy tiềm năng của việc phát triển cộng đồng công nghệ và bắt đầu ủng hộ, hỗ trợ tổ chức các hoạt động cộng đồng dần nhiều hơn.

Các meetup diễn ra thường xuyên hơn trong giai đoạn này, dù chưa thật sự “thoát ra” được những vấn đề bất cập của giai đoạn trước đó. Đây là thời điểm mà nhiều nhóm cộng đồng mạnh như Ruby Vietnam, Grokking... được hình thành và bắt đầu phát triển mạnh trong giai đoạn mới. Cùng lúc đó, **Dreamplex**, một co-working space ở phân khúc cao cấp cũng được thành lập trong giai đoạn này. **Dreamplex** đã hỗ trợ địa điểm cho khá nhiều hoạt động cộng đồng, với cơ sở vật chất tiện nghi, khang trang. Đây là một nguồn động lực cho khá nhiều người làm cộng đồng lúc bấy giờ.

2016: năm khởi nghiệp

Được chính phủ xem như là năm khởi nghiệp, năm 2016 chứng kiến nhiều cột mốc quan trọng của cộng đồng công nghệ ở Việt Nam. Các tuyên bố đầu tư, hỗ trợ hệ sinh thái khởi nghiệp được đưa ra, các diễn đàn, hội chợ được tổ chức. Các phương tiện truyền thông cũng góp phần đem những tin tức về thế giới công nghệ và startup tới mọi người một cách dồn dập và đầy hào hứng hơn. Cùng lúc đó, hoạt động cộng đồng cũng bắt đầu có sự thay đổi về chất mạnh mẽ hơn, khi một số tổ chức, cá nhân bắt đầu thực hiện những ý tưởng mới với kế hoạch phát triển dài hạn và có tính bền vững cao.

Trong số các công ty đã góp phần hỗ trợ cộng đồng trong giai đoạn này, Sillicon Straits Saigon được xem là một trong các nhân tố nổi bật khi đồng thời đứng ra tổ chức và cung cấp địa điểm cho các hoạt động của cộng đồng nói chung. Các hoạt động như Startup Weekend, Angel Hack đều được tổ chức ở đây. Ngoài ra, những mô hình mới như Geeky Weekend ra đời, đánh dấu thời điểm sôi động và bùng nổ của các cộng đồng công nghệ. Mỗi buổi meet-up của Geeky Weekend thường quy tụ từ 80 tới 200 lập trình viên với ước chừng ít nhất 30 người thường xuyên tham dự.

Đây cũng là năm bản thân Grokking định hình hướng phát triển và đi đến quyết định thành lập tổ chức phi lợi nhuận Grokking Vietnam.

Những bài học được rút ra

Với cơ hội được làm việc xuyên suốt với những cá nhân và tổ chức suốt từ giai đoạn đầu cho đến thời điểm hiện tại, người viết xin phép được chia sẻ một vài bài học được rút ra từ quan sát và kinh nghiệm thực tiễn, với hy vọng sẽ cung cấp cho bạn đọc góc nhìn gần gũi và rõ ràng hơn về những cộng đồng đang hoạt động cũng như các cộng đồng và hội nhóm vừa thành lập.



Một mục tiêu, sứ mệnh rõ ràng. Nếu những ai đã từng tham gia vào khâu tổ chức các hoạt động cộng đồng nói chung từ 1-2 năm trở lên, hẳn đều trải qua cảm giác này: bị mất phương hướng, cảm giác buồn chán và dễ bỏ cuộc. Vượt qua cái cảm giác hứng thú ban đầu, hầu hết công việc liên quan đến cộng đồng sẽ đòi hỏi tính nhẫn耐 và bền bỉ. Điều này cần được bắt nguồn từ sứ mệnh mà bản thân người tổ chức tự đặt ra cho mình. “Ăn cơm nhà, vác tù và hàng tổng” vốn dĩ chưa bao giờ là một công việc dễ dàng, nhưng chỉ cần có một sứ mệnh đủ mạnh, tự khắc động lực và sự bền bỉ sẽ tới.

Đảm bảo tính nhất quán cho nội dung. Bản thân người viết nhìn thấy nhiều hội nhóm được lập ra một cách liên tục. Có nhóm ban đầu là 50 người, sau đó lên vài trăm, sau đó cũng chìm dần. Quá trình chìm có khi trong vài tuần là thấy ngay, cũng có khi trong vài tháng đến một năm. Điểm chung của những nhóm thảo luận bị chìm này thường rơi vào tình huống: nội dung loãng do các thành viên chia sẻ nội dung không định hướng. Bản thân người tổ chức cộng đồng cần đảm bảo được tính nhất quán cho cộng đồng của mình. Những nội dung gì là nên chia sẻ và những nội dung là không nên chia sẻ.

Đòi hỏi sự chuẩn bị và tính chuyên nghiệp cao hơn. Khác với giai đoạn đầu, khi các hoạt động cộng đồng còn đang mạnh nha, nội dung còn sơ xài, dần dần, việc tổ chức các hoạt động cộng đồng sẽ đòi hỏi mang tính chỉnh chu và bài bản hơn. Đặc biệt, cần phân biệt rõ ràng giữa sự đơn giản, gọn gàng của một sự kiện (có được nhờ kinh nghiệm tổ chức, tối ưu hóa từng khâu chuẩn bị) với sự qua loa, sơ sài. Sự nhầm lẫn này rất dễ xảy ra, đặc biệt là đối với những hoạt động có vẻ dễ tổ chức như một buổi gặp gỡ kết nối quy mô nhỏ (coffee talk) đơn thuần.

Phát triển về lượng, hay là chết. Đứng lại, tức là thụt lùi. Điều này rất đúng, đặc biệt là cho lĩnh vực công nghệ nói chung và việc tổ chức cộng đồng nói riêng. Bản thân người viết đã từng tự hỏi: việc xây dựng một cộng đồng meetup ổn định về số lượng người tham gia liệu có khả thi hay không? Và kết luận bản thân tự rút ra đó là đối với cộng đồng: chất lượng nội dung và người tham dự phải đảm bảo, số lượng phải tăng dần. Số lượng tăng dần nhanh, hay chậm, còn tùy thuộc độ kiên nhẫn và tầm nhìn mà người tổ chức tự đặt ra cho mình. Nhưng song song với đó, chất lượng tốt chính là điều then chốt quyết định ý nghĩa của việc cộng đồng tồn tại. Nếu không đảm bảo được hai yếu tố này, bản thân cộng đồng sẽ dần dần co cụm lại thành một nhóm nhỏ những người quen biết, rồi dần dần sẽ mất hút.



Cởi mở trong việc ghi nhận ý kiến đóng góp từ cộng đồng, nhưng đừng lệ thuộc vào nó.

Xây dựng cộng đồng là một công việc mang tính khách quan, bạn sẽ phải thường xuyên lấy ý kiến đóng góp từ cộng đồng. Vậy đâu là điểm dừng? Theo người viết, bạn nên xác định rõ mình nên nghe ý kiến đóng góp từ ai. Một thực tế đáng buồn là góp ý thì dễ, nhưng chịu trách nhiệm với ý kiến đóng góp của bản thân mình thì khó. Khi bạn đặt câu hỏi: Cần phải làm gì trong lần hoạt động tiếp theo? Bạn dễ dàng nghe ý kiến đóng góp về ý tưởng. Nhưng hãy tự hỏi, liệu những người đề xuất ý tưởng có tham gia chính hoạt động đó không? Theo kinh nghiệm của cá nhân người viết, phần lớn câu trả lời là “không”. Hãy lắng nghe, hãy tôn trọng sự phản hồi, nhưng với một tâm thái có chọn lọc.

Hãy kiểm người đồng tổ chức. Giống như khởi nghiệp vậy, người đồng hành cùng bạn trong việc xây dựng cộng đồng là vô cùng quan trọng. Sẽ rất tốt nếu như người đồng tổ chức của bạn có những kỹ năng mà bạn còn thiếu hoặc còn yếu, vì đây sẽ là phần bổ sung quan trọng, định hình cho một tổ chức hoàn hảo hơn.

Và cuối cùng, hãy kiên nhẫn.

GÓC THỦ THÁCH

Đối với một người kỹ sư phần mềm, việc không ngừng ôn luyện kiến thức là điều không thể thiếu. Đến với chuyên mục Góc thử thách, mỗi kỳ team Grokking sẽ giới thiệu đến các bạn một vài câu đố nhỏ nhằm giúp các bạn có cơ hội review lại vốn kiến thức của mình.

Các bạn nào muốn thảo luận đáp án vui lòng ghé thăm diễn đàn thảo luận của Grokking nhé:
discuss.grokking.org

CÂU 1: [OS]

Parallelism & Concurrency, chọn tất cả câu sai:

- A. Nếu máy tính chỉ có 1 CPU core thì khi chạy chương trình sẽ không đạt được parallelism
- B. Nếu máy tính chỉ có 1 CPU core thì khi chạy chương trình sẽ không đạt được concurrency
- C. Parallelism chỉ đạt được khi chương trình chạy ở các processes khác nhau
- D. Khi viết parallel program sẽ không bị vấn đề về race condition

CÂU 2: [NETWORK]

Cho địa chỉ IP 162.16.168.196/28, hỏi IP 162.16.168.30 có nằm cùng đường mạng hay không?

- A. Có
- B. Không

CÂU 3: [OS]

Chỉ ra những tính chất của một process (Chọn tất cả những câu đúng):

- A. Bên trong 1 process có thể có nhiều thread
- B. Bên trong một thread có thể có nhiều process
- C. Các process không sử dụng chung một không gian vùng nhớ
- D. Các process giao tiếp với nhau qua IPC

CÂU 4: [DATABASE INDEX]

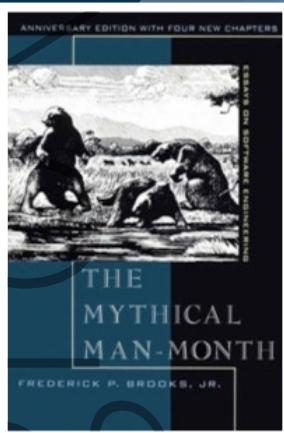
Chọn tất cả các câu đúng về database indexes trong PostgreSQL:

- A. Database index được biểu diễn bằng b-tree
- B. Khi thêm index vào, tốc độ truy xuất dữ liệu sẽ có khả năng nhanh hơn trong khi đó tốc độ insert/update không đổi
- C. Partial Index là kỹ thuật được sử dụng để đánh index trên một tập con dữ liệu trong bảng.
- D. Index (column_a, column_b) tương đương với index (column_b, column_a)
- E. B-Tree Index có thể sử dụng với những query như LIKE, SIMILAR TO

CÂU 5:

Chọn những câu đúng về docker:

- A. Lệnh docker inspect <container name> dùng để xem logs trong quá trình vận hành của một container.
- B. Lệnh docker ps sẽ liệt kê danh sách những container đã shutdown.
- C. Lệnh docker-compose run sẽ khởi động tất cả những containers.
- D. Tham số --ipaddress trong lệnh docker run sẽ gán địa chỉ IP cho container.
- E. Có thể map nhiều port bên trong các container với một port của server.



THE MYTHICAL MAN MONTH

Là cuốn sách kinh điển về quản lý phát triển phần mềm. Dù được viết vào khoảng năm 60-70 nhưng vẫn có nhiều bài học bổ ích cho đến tận hôm nay.

“Thêm người vào một dự án phần mềm đã bị trễ càng làm nó trễ hơn”, đây là một trong các ý được đề cập trong sách, nghe qua có vẻ ngược đời nhưng thực tế luôn như vậy. Dự án phần mềm thường phức tạp, trong đó việc trao đổi thông tin giữa những thành viên sẽ tốn khá nhiều thời gian. Chắc là bạn cũng đã từng tự hỏi là tại sao phải mất ngắn ấy thời gian chỉ để thảo luận, họp, giới thiệu việc cho người mới thay vì đơn giản là viết code?

Ngoài ra, trong sách còn đề cập đến những mô hình phát triển phần mềm mới như Agile Development, Extreme Programming giúp chia nhỏ công việc, chia nhỏ nhóm để tránh công kẽm, giúp quá trình triển khai dự án trở nên tinh gọn và thích nghi với sự thay đổi nhiều hơn.

Nếu công việc của bạn liên quan đến phát triển phần mềm, dù là lập trình viên hay là quản lý, đang làm việc cho dự án lớn hay dự án khởi nghiệp thì cuốn sách này cũng đều bổ ích.

BẢN CÓ BIẾT

■ 780.926

là tổng số nhân lực trong ngành công nghiệp CNTT tính đến năm 2016. Trong đó, **27,4%** là tỉ lệ lao động đang tham gia lĩnh vực công nghiệp phần mềm, công nghiệp nội dung số và dịch vụ CNTT.

■ Tổng số doanh nghiệp CNTT cả nước năm 2016 ước tính là **24.501** doanh nghiệp tăng **13,13%** so với năm 2015.

■ Tổng doanh thu lĩnh vực Công nghiệp CNTT năm 2016 ước tính đạt **1.500.009 tỷ đồng** (*tương đương 67,693 tỷ USD*) tăng **11,49%** so với năm 2015, trong đó công nghiệp phần cứng là **58.838 tỷ USD**, công nghiệp phần mềm là **3.038 tỷ USD**, công nghiệp nội dung số là **739 triệu USD** và dịch vụ CNTT (*trừ buôn bán, phân phối*) là **5.078 tỷ USD**.

■ Về đào tạo và phát triển nguồn nhân lực CNTT, số lượng các trường địa học, cao đẳng có đào tạo về CNTT, điện tử, viễn thông, an toàn thông tin năm 2016 là 250 trường, với tổng số chỉ tiêu tuyển sinh đại học, cao đẳng các ngành CNTT-TT là trên **68.000** sinh viên. Tuy nhiên, tỉ lệ thực tế tuyển sinh đại học, cao đẳng đạt **77,12%** và tỉ lệ tốt nghiệp đại học, cao đẳng ngành CNTT-TT đạt **93,88%**.

Đối với đào tạo nghề, cả nước có khoảng **164** trường cao đẳng nghề, trung cấp nghề đào tạo về CNTT, điện tử, viễn thông và an toàn thông tin, với tổng chỉ tiêu tuyển sinh là **18.311** sinh viên, tỉ lệ thực tế tuyển sinh đạt **68,27%** và tỉ lệ tốt nghiệp là **52,4%**.

(Theo Sách Trắng CNTT-TT Việt Nam năm 2017 do Bộ TT&TT phát hành)



In: 500 bản. Khổ 20.5 x 29.5 cm.

In tại: Công ty CP in Gia Định-

9D Nơ Trang Long, P. 7, Q. Bình Thạnh, TP. HCM.

Số ĐKKHXB: 660 - 2018/ CXBIPH / 1 - 38 / ĐoN, Cục Xuất bản,

In và phát hành xác nhận ngày: 01/03/2018,

Quyết định xuất bản số: 70B/QĐ-ĐoN, do NXB Đồng Nai cấp ngày
13/3/2018.

In xong và nộp lưu chiểu: quý 2/2018.

Nhà xuất bản Đồng Nai

1953J (210 cũ) Nguyễn Ái Quốc, TP Biên Hòa, Đồng Nai

Ban Biên tập: (02513) 825 292

P Kinh doanh: 946 521

P Kế toán: 946 520

Fax: (02513) 946 530 - Email: nxbdongnai@hcm.vnn.vn

The image consists of a repeating, abstract pattern of the word "grokking". The word is rendered in a bold, sans-serif font. It appears in different sizes, orientations, and positions across the frame. Some instances are rotated vertically or horizontally. The color of the text is a uniform, light gray, which creates a subtle, organic feel against the plain white background.

grokking
VIETNAM



NHÀ XUẤT BẢN ĐỒNG NAI

Giá: 50.000 VNĐ

